ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ НИЖЕГОРОДСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО»

Отчет по лабораторной работе №3 по дисциплине «Алгоритмы и структуры данных»

Выполнил:

Студент Ненев А.Е.

Группа

3824Б1ФИ2

Нижний Новгород

2025г

1. Описание программной реализации

Программа реализует вычислитель арифметических выражений с поддержкой переменных, унарных и бинарных операторов, скобок и функции синус. Реализация основана на преобразовании инфиксной записи выражения в обратную польскую запись с последующим вычислением.

2. Описание классов, методов и функций

1) Структура Token

Назначение: Хранение токена (лексемы) с его позицией в исходной строке.

Поля:

value (std::string) - значение токена

start_pos (size_t) - начальная позиция в строке (нумерация с 1)

end_pos (size_t) - конечная позиция в строке

2) Класс ExpressionError

Назначение: Пользовательское исключение для ошибок в выражении с хранением позиций ошибок.

Родительский класс: std::runtime_error

Конструктор:

ExpressionError(const std::string& msg, std::vector<size_t> positions)

Параметры:

msg - сообщение об ошибке

positions - вектор позиций, связанных с ошибкой

Методы:

1. **const std::vector<size_t>& getPositions() const**

Функционал: Возвращает вектор позиций ошибок

3) Шаблонный класс TStack<T>

Назначение: Реализация стека на основе std::vector.

Методы:

1. **bool empty() const**

Функционал: Возвращает true, если стек пуст

2. **size_t size() const**

Функционал: Возвращает количество элементов в стеке

3. **void push(const T& element)**

Функционал: Добавляет элемент в стек

Параметры:

element - добавляемый элемент

Сложность: O(1) в среднем, O(n) в худшем случае (при перевыделении памяти)

4. **void pop()**

Функционал: Удаляет верхний элемент стека

Генерирует исключение std::out_of_range при пустом стеке

5. **T& top()**

Функционал: Возвращает ссылку на верхний элемент стека

Генерирует исключение std::out_of_range при пустом стеке

6. **const T& top() const**

Функционал: Константная версия метода top()

4) Класс TPostfix

Назначение: Основной класс для преобразования и вычисления выражений.

Поля:

variables (std::map<std::string, double>) - хранилище значений переменных

Приватные методы:

1. **bool isBinaryOperator(const std::string& token) const**

Функционал: Проверяет, является ли токен бинарным оператором (+, -, *, /)

Параметры:

token - проверяемый токен

Возвращает true, если токен - бинарный оператор

2. **bool isUnaryOperator(const std::string& token) const**

Функционал: Проверяет, является ли токен унарным оператором (u-)

Параметры:

token - проверяемый токен

Возвращает true, если токен - унарный оператор

3. **bool isOperator(const std::string& token) const**

Функционал: Проверяет, является ли токен оператором (бинарным или унарным)

Параметры:

token - проверяемый токен

Возвращает true, если токен - оператор

4. **bool isFunction(const std::string& token) const**

Функционал: Проверяет, является ли токен функцией (sin)

Параметры:

token - проверяемый токен

Возвращает true, если токен - функция

5. **bool isNumber(const std::string& token) const**

Функционал: Проверяет, является ли токен числом

Параметры:

token - проверяемый токен

Возвращает true, если токен - число

Сложность: O(n), где n - длина токена (из-за strtod)

6. **bool isVariable(const std::string& token) const**

Функционал: Проверяет, является ли токен переменной (т.е. состоит только из букв)

Параметры:

token - проверяемый токен

Возвращает true, если токен - переменная

Сложность: O(n), где n - длина токена (из-за std::all_of)

7. **int getPrecedence(const std::string& op) const**

Функционал: Возвращает приоритет оператора

Параметры:

op - оператор

Возвращает: 3 для унарных операторов и функций, 2 для */, 1 для +-, 0 для остальных

8. **bool isLeftAssociative(const std::string& op) const**

Функционал: Проверяет левоассоциативность оператора

Параметры:

op - оператор

Возвращает true для бинарных операторов, false для унарных

9. **void validateTokenSequence(const std::vector<Token>& tokens)**

Функционал: Проверяет корректность последовательности токенов

Параметры:

tokens - вектор токенов для проверки

Генерирует ExpressionError при обнаружении ошибок

Сложность: O(n), где n - количество токенов

10. **std::vector<Token> processUnaryMinus(const std::vector<Token>& tokens)**

Функционал: Заменяет унарные минусы на специальный токен "u-"

Параметры:

tokens - исходный вектор токенов

Возвращает модифицированный вектор токенов

Сложность: O(n), где n - количество токенов

11. **std::vector<Token> toPostfix(const std::vector<Token>& infixTokens)**

Функционал: Преобразует инфиксную запись в постфиксную

Параметры:

infixTokens - токены в инфиксной записи

Возвращает токены в постфиксной записи

Сложность: O(n), где n - количество токенов

12. **double evaluatePostfix(const std::vector<Token>& postfixTokens)**

Функционал: Вычисляет значение постфиксного выражения

Параметры:

postfixTokens - токены в постфиксной записи

Возвращает результат вычисления

Сложность: O(n), где n - количество токенов

13.std::vector<Token> tokenize(const std::string& expression)

Функционал: Разбивает строку на токены

Параметры:

expression - входная строка выражения

Возвращает вектор токенов

Сложность: O(n), где n - длина строки

Публичные методы:

14.double evaluate(const std::string& expression)

Функционал: Основной метод вычисления выражения

Параметры:

expression - строка с выражением

Возвращает результат вычисления

Общая сложность: O(n), где n - длина входной строки

15.void setVariable(const std::string& name, double value)

Функционал: Устанавливает значение переменной

Параметры:

name - имя переменной

value - значение переменной

Сложность: O(log m) для вставки в std::map, где m - количество переменных

16.bool hasVariable(const std::string& name) const

Функционал: Проверяет наличие переменной

Параметры:

name - имя переменной

Возвращает true, если переменная существует

Сложность: O(log m), где m - количество переменных

17.void clearVariables()

Функционал: Очищает все сохраненные значения переменных

Сложность: O(m), где m - количество переменных

Краткие комментарии к тестам

Тесты класса TStack

TestNameStackNoExcept

Проверяет корректную работу всех основных операций стека без генерации исключений.

TestNameStackCheckExceptions

Проверяет генерацию исключений при попытке доступа к элементам или удаления из пустого стека.

Тесты класса TPostfix

TestNameOperandsOnly

Проверяет вычисление выражений, состоящих только из одного числа, а также обработку пустой строки (должно генерировать исключение).

TestNameVariablesOnly

Проверяет работу с одиночными переменными и унарным минусом перед переменной.

TestNameTPostfixNoExeptIntegersSimple

Тестирует простые арифметические операции с целыми числами (сложение, вычитание, умножение, деление) без генерации исключений.

TestNameTPostfixNoExeptFloatsSimple

Тестирует простые арифметические операции с числами с плавающей точкой, проверяя точность вычислений без генерации исключений.

TestNameTPostfixNoExeptIntegersComplicated

Проверяет сложные выражения с целыми числами, включая комбинации операторов и унарный минус, не бросает исключения.

TestNameTPostfixNoExeptFloatsComplicated

Проверяет сложные выражения с числами с плавающей точкой, включая множественные операции, не бросает исключения.

TestNameTPostfixNoExceptRoundBrackets

Тестирует корректность обработки скобок и приоритета операций внутри скобок.

TestNameTPostfixExceptions

Комплексная проверка обработки некорректных выражений:

Операторы в начале/конце выражения

Двойные операторы подряд

Несбалансированные скобки

Недопустимые символы

Деление на ноль в различных контекстах

TestNameVariablesNoExcept

Проверяет корректную работу с переменными в различных выражениях, включая очистку переменных между тестами, не бросает исключения.

TestCaseNameVariablesExceptions

Тестирует обработку ошибок при работе с переменными:

Некорректный синтаксис с переменными

Смешанные буквенно-цифровые токены

Деление на переменную со значением ноль

TestCaseNameSinus

Проверяет работу функции синуса:

Корректное вычисление sin() с переменными и числами

Вложенные вызовы sin(sin(sin()))

Ошибки при некорректном синтаксисе функции – отсутствии аргумента или его запись без скобок

TestCaseNameUnaryMinusCheck

Тестирует обработку унарного минуса:

Выражения с переменными

Двойное отрицание -(-4)

Вычитание отрицательного числа (5--4)

Ошибка при двойном унарном минусе (--4)

TestCaseNameSameVariables

Проверяет корректность вычисления выражений с отрицательными числами и унарными операторами в различных комбинациях.

TestCaseNamePriorityCheck

Проверяет соблюдение приоритета операций (умножение/деление выполняется перед сложением/вычитанием).