

Федеральное государственное автономное образовательное учреждение  
высшего образования "Национальный исследовательский Нижегородский  
государственный университет им. Н.И. Лобачевского"

**Отчёт по лабораторной работе №3**  
**дисциплины «Алгоритмы и структуры данных»**

Выполнил:

Студент группы 3824Б1ФИ2

Старостин Д.Д.

Нижний Новгород – 2025 г.

# **1. Постановка задачи**

Цель данной работы – разработка структуры данных Стек и ее использование для вычисления арифметических выражений. Выражение в качестве операндов может содержать переменные и вещественные числа. Допустимые операции: +, -, /, \*, унарный -, математическая функция cos. Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю вид ошибки и номера символов строки, в которых были найдены ошибки. При вычислении арифметического выражения требуется ввод с консоли неизвестных переменных.

В ходе выполнения работы должны быть решены следующие задачи:

1. Разработка интерфейса шаблонного класса TStack.
2. Реализация методов шаблонного класса TStack.
3. Разработка интерфейса TPostfix для вычисления арифметических выражений с использованием обратной польской записи.
4. Реализация методов класса TPostfix.
5. Написание и обеспечение работоспособности тестов, покрывающих основной функционал и краевые случаи.

## **2. Описание программной реализации**

### **2.1.1. Описание класса TStack**

Шаблонный класс TStack реализует структуру данных stack, работающую по принципу LIFO, с минимальным необходимым функционалом для применения в классе TPostfix. Содержит поля:

- int top\_ – индекс верхнего элемента в стеке; значение -1 означает, что стек пуст;
- std::vector<T> data – динамический массив, в котором хранятся элементы стека;

Методы класса позволяют узнать текущий размер стека, получить значение верхнего элемента, узнать, пуст ли стек, а также извлечь и добавить элемент в стек.

### **2.1.2 Описание методов класса и функций**

- 1) TStack() – конструктор по умолчанию.
  - Параметры и возвращаемые значения: –

- Функционал: инициализирует пустой стек, присваивая полю `top_` значение `-1`.
- 2) `size_t size() const` – получение размера стека.
- Параметры и возвращаемые значения: возвращает количество элементов в стеке.
  - Функционал: вычисляет размер стека на основе значения поля `top_`.
- 3) `bool empty() const` – проверка пустоты стека.
- Параметры и возвращаемые значения: возвращает `true`, если стек пуст, `false` – иначе.
  - Функционал: позволяет узнать, пуст ли стек.
- 4) `void push(const T& val)` – добавление элемента.
- Параметры и возвращаемые значения: принимает `val` – константную ссылку на вставляемый в стек элемент.
  - Функционал: добавляет в стек элемент `val` и увеличивает `top_` на 1.
- 5) `T top() const` – получение верхнего элемента.
- Параметры и возвращаемые значения: возвращает значение верхнего элемента стека.
  - Функционал: предоставляет доступ к значению верхнего элемента стека.
- 6) `void pop()` – удаление верхнего элемента.
- Параметры и возвращаемые значения: –
  - Функционал: удаляет верхний элемент из стека и уменьшает `top_` на 1.

## 2.2.1. Описание класса `TPostfix`

Класс `TPostfix` реализует интерфейс для вычисления арифметических выражений с использованием обратной польской записи.

Поля:

- std::string infix – инфиксная запись выражения.
- std::string postfix – постфиксная запись выражения.
- std::vector<std::string> postfix\_lexems – постфиксная запись, разбитая на лексемы для дальнейшей обработки.
- std::vector<std::string> lexems – инфиксная запись, разбитая на лексемы для дальнейшей обработки.
- std::vector<std::pair<std::string, double>> operands – список operandов, хранящийся в векторе.
- const std::vector<std::pair<char, int>> priorities – список приоритетов операций, хранящийся в векторе.

Приватные методы класса проверяют корректность введённого выражения, переводят его в постфиксную форму.

Публичные методы класса позволяют получить выражение в инфиксной или постфиксной форме, operandы, которые представлены в введённом выражении, а также посчитать значение введённого выражения с переменными, значения которых пользователь либо передаёт в метод Calculate, либо вводит через консоль во время исполнения программы.

### **2.2.2. Описание методов класса и функций**

- 1) TPostfix(const std::string& src) – конструктор принимающий выражение в виде строки.
  - Параметры и возвращаемые значения: src – строка содержащая выражение.
  - Функционал: инициализирует список операций и их приоритетов, присваивает полю infix исходное выражение. Вызывает методы, преобразующие исходные выражения в постфиксную форму.
  - Сложность: O(n). Вызывает ToPostfix() (O(n)), который вызывает Parse() (O(n)).
- 2) void Parse() – первичная обработка исходного выражения.
  - Параметры и возвращаемые значения: –
  - Функционал: проверяет исходное выражение на корректность разбивает его на лексемы, заполняя поле lexems, и находит operandы, заполняя поле operands. При обнаружении ошибок во введённой строке, выдаёт ошибку с указанием места ошибки.

- Сложность:  $O(n)$ .  $n$  – размер строки с исходным выражением. Обрабатывается каждый символ в цикле.
- 3) void ToPostfix() – преобразование выражения в постфиксную форму.
- Параметры и возвращаемые значения: –
  - Функционал: вызывает метод Parse(), который разбивает исходное выражение на лексемы, и находит операнды. Далее преобразует выражение в постфиксную форму.
  - Сложность:  $O(n)$ . Метод Parse() –  $O(n)$ . Цикл по lexems –  $O(n)$ ,  $n$  – количество лексем. Конечный while –  $O(n)$ . Наличие внутреннего цикла по priorities не влияет на линейную сложность, так как размер priorities фиксированный. Таким образом, складывая сложности всех частей получаем линейную сложность.
- 4) std::string GetInfix() const – получение выражения в инфиксной форме.
- Параметры и возвращаемые значения: возвращает строку – выражение в инфиксной форме.
  - Функционал: возвращает поле infix.
- 5) std::string GetPostfix() const – получение выражения в постфиксной форме.
- Параметры и возвращаемые значения: возвращает строку – выражение в постфиксной форме.
  - Функционал: возвращает поле postfix.
- 6) std::vector<std::string> GetOperands() const – получение списка operandов.
- Параметры и возвращаемые значения: возвращает вектор ( $\text{std}::\text{vector}<\text{std}::\text{string}>$ ), содержащий operandы, в виде строк ( $\text{std}::\text{string}$ )
  - Функционал: используя поле operands формирует список operandов и возвращает его.
  - Сложность:  $O(m)$ .  $m$  – количество operandов. Формирование списка operandов происходит в цикле с  $m$  итерациями.

- 7) double Calculate() – Вычисление значения выражения с использованием значений операндов, введённым пользователем в консоль.
- Параметры и возвращаемые значения: возвращает значение выражения.
  - Функционал: считывает значения операндов из консоли. Вычисляет значение выражения используя поле `postfix_lexems` и введённые значения операндов.
  - Сложность:  $n$  – количество лексем в постфиксной записи,  $m$  – количество операндов. Ввод операндов –  $O(m)$ . Цикл по всем лексемам –  $n$  итераций ( $O(n)$ ). Если лексема – операнд, то выполняется вложенный цикл поиска операнда в массиве операндов –  $O(m)$ . Итоговая сложность:  $O(m) + n * O(m) = O(n * m)$ .
- 8) double Calculate(const std::vector<std::pair<std::string, double>>& values) – Вычисление значения выражения с использованием значений операндов, переданных пользователем в функцию.
- Параметры и возвращаемые значения: принимает вектор пар и названий операндов и их значений. Возвращает значение выражения.
  - Функционал: выдаёт ошибку, если передано недостаточно операндов, или какой-либо operand отсутствует. Присваивает operandам переданные значения. Вычисляет значение выражения используя поле `postfix_lexems` и переданные значения операндов.
  - Сложность:  $n$  – количество лексем в постфиксной записи,  $m$  – количество операндов в выражении,  $k$  – количество переданных операндов ( $\geq$  числа операндов в выражении  $m$ ). Присваивание переданным значений operandам:  $k * O(m)$ . В каждой итерации цикла по массиву переданных operandов и их значений ( $k$  итераций) происходит поиск в массиве `operands` ( $O(m)$ ) для сопоставления operandu из выражения значения из переданного массива. Цикл по всем лексемам –  $n$  итераций ( $O(n)$ ). Если лексема – операнд, то выполняется вложенный цикл поиска операнда в массиве operandов –  $O(m)$ . Итоговая сложность:  $O(k * m) + O(n * m) = O(m * (n + k))$ .

### 3. Тесты класса TPostfix

- 1) Блок Constructor:

- 1.1) one\_operation\_no\_throw: проверка корректной работы без выброса исключений конструктора при передаче простых выражений с одной операцией.
- 1.2) two\_unar\_operators\_in\_row: проверка корректной работы без выброса исключений конструктора при передаче простых выражений с вложенными унарными операциями.
- 1.3) complex\_formulas: проверка корректной работы без выброса исключений конструктора при передаче более сложных выражений, содержащих множество разных операций, operandов и чисел-operandов в разных сочетаниях.

2) Тест GetInfix:

- is\_correct: Проверка корректности работы метода GetInfix.

3) Тест GetPostfix:

- is\_correct: Проверка корректности работы метода GetPostfix на сложных и простых выражениях.

4) Тест GetOperands:

- is\_correct: Проверка корректности работы метода GetOperands на выражения содержащих только уникальные operandы и на выражения с повторяющимися operandами.

5) Блок Calculate:

- 5.1) is\_correct\_and\_no\_throw: проверка корректной работы без выброса исключений метода Calculate с передачей значений operandов в функцию при передаче корректных выражений и данных для вычисления, то есть данных, в которых представлены как минимум все operandы, присутствующие в выражении.

- 5.2) throw\_: проверка того, что метод выбрасывает исключение при передаче некорректных данных.

6) Блок Parse:

- 6.1) brackets\_throw: проверка возникновения ошибок при попытке передать в конструктор выражения с некорректными скобочными

последовательностями и некорректными выражениями со скобками.

- 6.2) `variables_throw`: проверка возникновения ошибок при попытке передать в конструктор выражения с некорректным взаимным расположением операндов.
- 6.3) `digits_throw`: проверка возникновения ошибок при попытке передать в конструктор выражения с некорректным взаимным расположением чисел.
- 6.4) `point_throw`: проверка возникновения ошибок при попытке передать в конструктор выражение с некорректным числом (число с более чем одной плавающей точкой).
- 6.5) `unar_minus_throw`: проверка возникновения ошибок при попытке передать в конструктор выражений с некорректным использованием унарного минуса (более одного унарного минуса подряд).
- 6.6) `operations_throw`: проверка возникновения ошибок при попытке передать в конструктор выражений с некорректным взаимным расположением операций.
- 6.7) `unknown_symbol`: проверка возникновения ошибок при попытке передать в конструктор выражений с символом, не входящим в список допустимых операций, и не являющимся числом или операндом.