Министерство образования и науки 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

«Вычисление арифметических выражений»

Выполнил:
студент группы 3822Б1ПМ1
Сомов Я.В.

Проверил:
Волокитин В.Д.

Нижний Новгород
2023

Содержание

Введение

Основная цель работы — разработка структуры данных стек для дальнейшего её использования в программе, вычисляющей значение введённого пользователем арифметического выражения. Арифметическое выражение — это комбинация чисел, букв, символов функций и операций.

1. Постановка задачи

Для достижения поставленной цели необходимо решить следующие задачи:

- Разработать структуру данных стек;
- Разработать алгоритм лексического анализа введённого выражения;
- Разработать алгоритм проверки введённого выражения на правильность;
- Реализовать алгоритм перевода выражения из инфиксной формы в постфиксную;
- Реализовать алгоритм вычисления выражения по постфиксной записи.

Для простоты реализации итоговой программы предполагается, что для составления арифметического выражения используются только числа, записанные в экспоненциальной форме или в форме десятичной дроби, пользовательские переменные, заданные в виде комбинаций букв, бинарные (сложение, вычитание, умножение, деление, возведение в степень) и унарные (унарный минус) операции и некоторые наиболее часто используемые математические функции.

2. Руководство пользователя

Программа осуществляет работу с пользователем в режиме интерфейса командной строки.

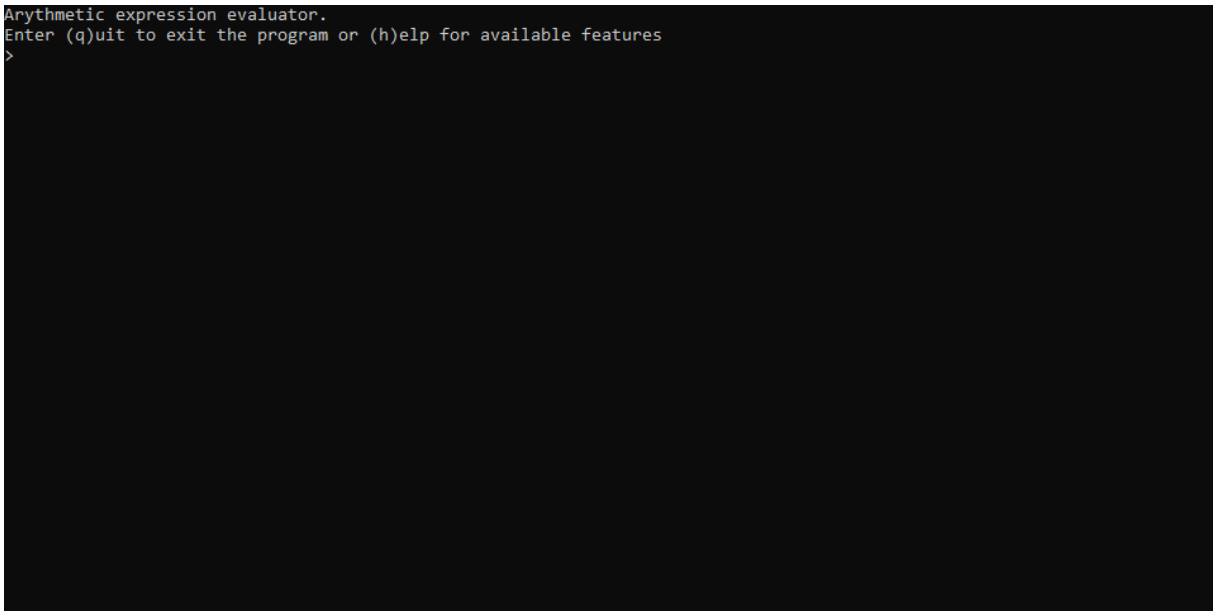

Рисунок 1. Внешний вид интерфейса программы при запуске.

Пользователь может выйти в любой момент из программы, написав q или quit. Для получения списка доступных возможностей пользователь может написать h или help.

Работа с программой представляет собой последовательный ввод арифметических выражений. Программа выведет сообщение об ошибке в случае, если при вводе была допущена ошибка или была введена строка, содержащая только пробельные символы. Если введённое выражение было корректным, программа выведет результат вычисленного выражения.

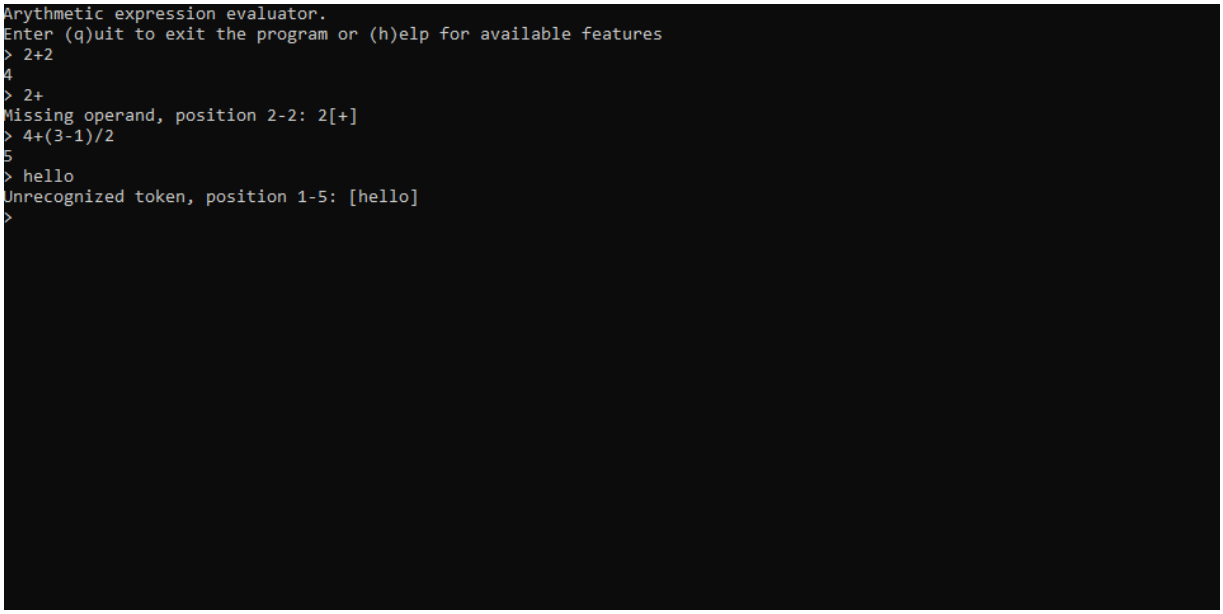

Рисунок 2. Сеанс работы с программой.

Программа способна обрабатывать:

- ввод численных констант в форме десятичной дроби или в экспоненциальной дроби;
- ввод математических констант e, π;
- ввод бинарных операций + (сложение), – (вычитание), * (умножение), / (деление), ^ (возведение в степень);
- ввод функций sin (синус), cos (косинус), tan (тангенс), asin (арксинус), acos (арккосинус), atan (арктангенс), exp (экспонента e^x), ln (натуральный логарифм), sqrt (квадратный корень);
- ввод пользовательских переменных.

Ввод пользовательских переменных осуществляется с помощью выражений вида [название переменной] = [значение] или [название переменной] = [выражение]. Допускается множественное присваивание вида [название переменной 1] = … = [название переменной n] = [значение] или [название переменной 1] = … = [название переменной n] = [выражение]. Не допускается перезаписывание констант e, π, попытка присваивания в выражение, использование невалидных названий переменных (т.е. названий математических функций и системных команд, названий, начинающихся с цифры, содержащих нелатинские буквы, знаки пунктуации, спецсимволы).

Для просмотра имеющихся переменных можно использовать команду vars. Для удаления всех пользовательских переменных можно использовать команду clear.

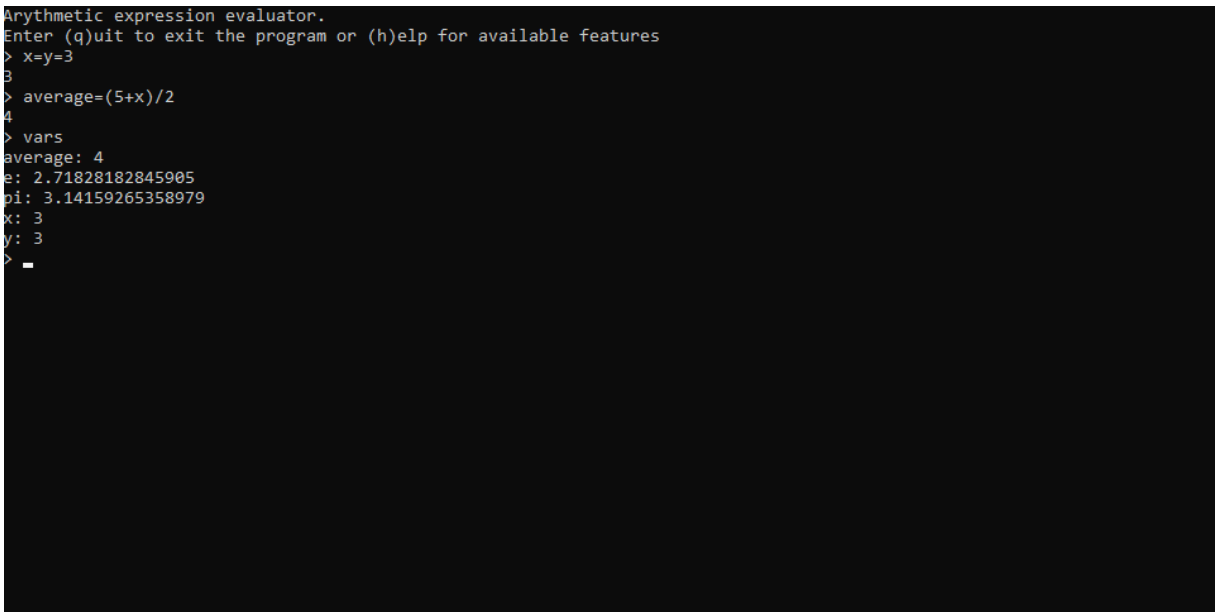

Рисунок 3. Пример работы с пользовательскими переменными.

3. Руководство программиста

3.1. Описание структуры программы

Структура проекта:

- gtest — файлы библиотеки Google Test.
- include
  - arithmetic.h — заголовочный файл, содержащий прототипы методов классов TPostfix, Parser, Tokenizer;
  - stack.h — заголовочный класс, содержащий реализацию шаблонного класса TStack;
- samples
  - CMakeLists.txt — файл для сборки пользовательского приложения
  - main_arithmetic.cpp — исходный файл, содержащий реализацию пользовательского приложения для вычисления значений арифметического выражения;
- sln — каталог с файлами решений для Microsoft Visual Studio.
- src
  - arithmetic.cpp — исходный файл, содержащий реализацию методов классов TPostfix, Parser, Tokenizer.
- test
  - CMakeLists.txt — файл для сборки тестового приложения.
  - test_arithmetic.cpp — исходный файл, содержащий реализацию модульных тестов для класса TPostfix.
  - test_stack.cpp — исходный файл, содержащий реализацию модульных тестов для класса TStack.
  - test_arithmetic.cpp — исходный файл программы, выполняющей модульное тестирование.
- .gitignore — перечень игнорируемых Git файлов.
- CMakeLists.txt — файл для сборки проекта CMake.
- README.md — файл с описанием целей и задач лабораторной работы.

- report.docx — файл отчёта по лабораторной работе (для работы в Microsot Word).
- report.pdf — файл отчёта по лабораторной работе (для чтения).

На рисунке №4 представлена диаграмма структуры класса TPostfix:

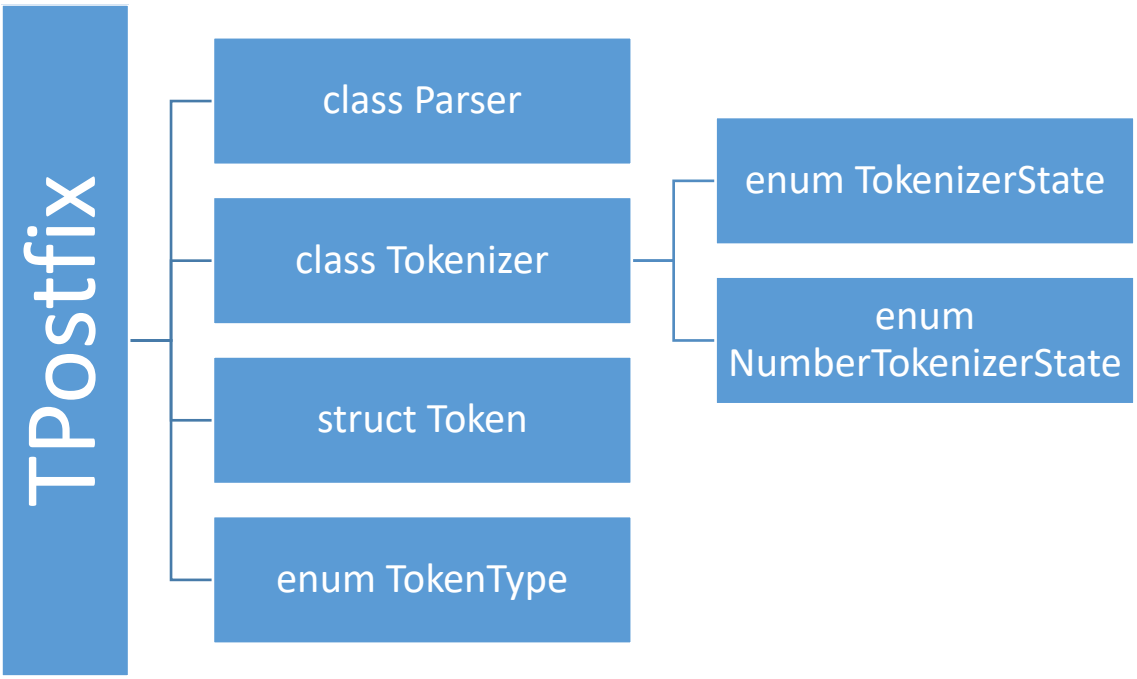

Рисунок 4. Структура класса TPostfix

class Parser отвечает за преобразование инфиксной формы выражения в постфиксную.

class Tokenizer отвечает за разбиение входного арифметического выражения, записанного в виде строки, в массив токенов.

struct Token реализует представление токенов в памяти. Включает в себя std::string s (строка токена), TokenType type (тип токена), double val (численное значение для переменных и численных констант, заполняется NaN для всех остальных типов токенов).

enum TokenType используется для реализации классификации типов токенов:

- унарная операция;
- бинарная операция;
- функция;
- число;
- переменная;
- операция присваивания;

- левые круглые скобки;
- правые круглые скобки;
- нераспознанный.

enum TokenizerState используется для хранения состояний конечного автомата, отвечающего за обработку входного строки с выражением и распознания числа, enum NumberTokenizerState — для хранения состояний конечного автомата, отвечающего за распознание чисел во входной строке.

3.2. Описание алгоритмов

В таблицах 1, 2 представлены прототипы реализованных функций с описанием их назначения.

Таблица 1. arithmetic.cpp

Прототип функции	Описание
`void throw_error(const std::string& s, const std::string& err_s, const size_t pos)`	Форматирует аргумент для std::runtime_error (строка с указанием позиции ошибочного символа).
`void throw_error(const std::string& s, const std::string& err_s, const std::string& t, const size_t pos)`	Форматирует аргумент для std::runtime_error (строка с указанием позиции ошибочного токена).
`TPostfix::TPostfix()`	Конструктор по умолчанию для класса TPostfix.
`double TPostfix::evaluate(const std::string& s)`	Вычисляет значение арифметического выражения s, переданного в виде строки. Возвращает double с вычисленным значением арифметического выражения. Выбрасывает исключение std::runtime_error в случае возникновения ошибки.
`const std::map<std::string, double>& TPostfix::getVariables() noexcept`	Возвращает константную ссылку на map с пользовательскими переменными.
`void TPostfix::clearVariables()`	Удаляет все пользовательские переменные.
`TPostfix::Token* TPostfix::Tokenizer::tokenize(const std::string&`	Разбивает введённую строку s, содержащее арифметическое выражение, записанное в инфиксной

`s, std::map<std::string, double>& vars, size_t& sz)`	форме, на токены. Возвращает массив токенов и число распознанных токенов (изменяет переданный по ссылке аргумент sz). Выбрасывает исключение std::runtime_error, если не был распознан токен, была передана пустая строка или строка, состоящая только из пробельных символов, обнаружены два токена, которые не могут стоять рядом друг с другом или обнаружено нераспознанное сочетание символов.
`TPostfix::Token TPostfix::Tokenizer::tokenizeNumber(const std::string& s, size_t& i)`	Распознаёт в строке s число на позиции i. Возвращает токен распознанного числа. Выбрасывает исключение std::runtime_error, если возникла ошибка в ходе распознания числа.
`TPostfix::Token TPostfix::Tokenizer::tokenizeOperation(const std::string& s, size_t& i, bool unary)`	Распознаёт в строке s унарную или бинарную операцию. Возвращает токен распознанной операции. Выбрасывает исключение std::runtime_error, если отсутствует операнд для бинарной операции.
`TPostfix::Token TPostfix::Tokenizer::tokenizeWord(const std::string& s, size_t& i)`	Распознаёт в строке s сочетание символов на позиции i. Возвращает токен с типом «функция» или с временно обозначенным типом «нераспознанный».
`inline bool TPostfix::Tokenizer::isInvalidVariableName(const std::string& s) noexcept`	Возвращает true, если название переменной — зарезервированная постоянная (e или π), название функции, бинарная или унарная операция, круглая скобка, название системной команды, пробельный символ, либо если название содержит символы, отличные от цифр и букв латинского алфавита. Возвращает false во всех остальных случаях.
`TPostfix::Token* TPostfix::Parser::convertToPostfix(Token* tokens, size_t& sz)`	Принимает массив токенов выражения, записанного в инфиксной форме, и его размер. Возвращает массив токенов выражения, записанного в постфиксной форме, и его размер без учёта скобок.
`inline int TPostfix::Parser::operatorPriority(const Token& t) noexcept`	Возвращает приоритет операции в виде целого числа (-1 для токенов, не являющихся операциями, 1 для

	присваивания, 2 для сложения, 3 для умножения и деления, 4 для возведения в степень и унарных операций, 5 для функций).
`inline bool TPostfix::Parser::isLeftAssoc(const Token& t) noexcept`	Возвращает true, если результат операции вычисляется слева направо, false, если справа налево.

Таблица 2. stack.h

Прототип функции	Описание
`TStack()`	Конструктор по умолчанию для класса TStack.
`TStack(const TStack& s)`	Конструктор копирования для класса TStack.
`TStack(TStack&& s)`	Конструктор перемещения для класса TStack.
`~TStack()`	Деструктор для класса TStack.
`TStack& operator=(const TStack& s)`	Копирующее присваивание для класса TStack.
`TStack& operator=(TStack&& s) noexcept`	Перемещающее присваивание для класса TStack.
`friend void swap(TStack& lhs, TStack& rhs) noexcept`	Реализация функции swap для класса TStack.
`void push(const T& value)`	Помещает данный элемент value в стек.
`void push(T&& value)`	Помещает данный элемент value в стек.
`void pop()`	Удаляет элемент с вершины стека.
`void clear()`	Удаляет все элементы из стека и освобождает занятую память.
`T& top()`	Возвращает ссылку на элемент с вершины стека.
`const T& top() const`	Возвращает константную ссылку на элемент с вершины стека.
`bool isEmpty()`	Возвращает true, если стек пуст, false в противном случае.
`size_t size()`	Возвращает число помещённых в стек элементов.
`void resize()`	Приватный метод класса TStack, перевыделяющий память в случаях, когда выделенной памяти не хватает для добавления в стек новых элементов.

Для преобразования строки в массив токенов применяется следующий алгоритм:

- Создать временный стек токенов, стек позиций скобок, положить, что может стоять унарная операция.
- Для каждого символа во входной строке
  - Если символ: пробел, передвинуться на символ вперёд.
  - Если символ: число или точка:
    - кинуть исключение, если последний токен — число, переменная или правая скобка
    - создать токен с типом «число»
    - пометить, что после токена не может стоять унарная операция
    - поместить токен во временный стек токенов.
  - Если символ +, –, /, *:
    - если на данном месте может стоять унарная операция
      - создать токен с типом «унарная операция»
      - если символ отличен от –, кинуть исключение.
      - кинуть исключение, если символ отличен от –, последний токен не унарная операция, не бинарная операция, не функция, не левая скобка, не присваивание
    - иначе
      - создать токен с типом «бинарная операция»
      - кинуть исключение, если последний токен не число, не переменная, не правая скобка, не присваивание
    - пометить, что после токена может стоять унарная операция
    - поместить токен во временный стек токенов.
  - Если символ (:

        - кинуть исключение, если последний токен не левая скобка, не унарная операция, не бинарная операция, не функция или не присваивание
        - поместить токен во временный стек токенов
        - поместить позицию скобки в стек позиций скобок
        - пометить, что после токена может стоять унарная операция
    - Если символ):
        - кинуть исключение, если стек позиций скобок пуст, последний токен не число или не переменная
        - поместить токен во временный стек токенов
        - изъять элемент из стека позиций скобок
        - пометить, что после токена не может стоять унарная операция
    - Если символ =:
        - кинуть исключение, если = первый символ, последний токен не переменная, или не нераспознанный, или происходит попытка присвоить в выражение
        - если последний токен имеет тип «нераспознанный», изменить его тип на «переменная»
        - создать токен с типом «операция присваивания»
        - поместить токен во временный стек токенов
        - пометить, что после токена может стоять унарная операция
    - Если символ — буква латинского алфавита:
        - выделить символы до первого встреченного пробела, унарной или бинарной операции, круглой скобки
        - если символы образуют слово, соответствующее какой-либо математической функции:
            - создать токен с типом «функция»
            - пометить, что после токена может стоять унарная операция

            - иначе, если символы образуют название существующей переменной:
                - создать токен с типом «переменная»
                - пометить, что после токена не может стоять унарная операция
            - иначе:
                - создать токен с типом «нераспознанный»
                - пометить, что после токена не может стоять унарная операция
            - кинуть исключение, если последний токен — не унарная операция, не бинарная операция, не функция, не левая скобка
            - поместить токен во временный стек токенов
        - Иначе: кинуть исключение
- Если нет токенов во временном стеке токенов, кинуть исключение (строка состояла только из пробелов)
- Если стек с позициями скобок не пуст, кинуть исключение (есть по крайней мере одна незакрытая скобка)
- Создать массив токенов, пока временный стек токенов не пуст:
    - Взять токен с вершины стека;
    - Если токен — нераспознанная последовательность символов: кинуть исключение
    - Иначе переложить токен из стека в массив

На выходе получается массив распознанных токенов.

Для распознания чисел применяется следующий алгоритм:

- Создать стеки цифр для целой и дробной частей десятичной дроби и для порядка числа
- Пока не достигнут конец или не прервано распознавание, двигаться по состояниям:
    - Состояние «выделить целую часть»

        - Если первый символ точка или ноль: целая часть равна нулю, перейти к выделению дробной части.
        - Помещать символы в стек для целой части пока текущий символ цифра или не точка.
        - Прервать распознание если встретился любой другой символ.
    - Состояние «выделить дробную часть»
        - Помещать символы в стек для дробной части пока текущий символ цифра или не e.
        - Прервать распознание если встретился любой другой символ.
    - Состояние «определить знак порядка»
        - Кинуть исключение, если встреченные символы не e+ или e–.
    - Состояние «выделить порядок»
        - Помещать символы в стек для порядка пока текущий символ цифра. Кинуть исключение если встретилась точка.
        - Прервать распознание если встретился любой другой символ.
    - Кинуть исключение, если длина токена единица и единственный символ — точка, или если распознание завершилось на стадии «определить знак порядка»
- Перевернуть стек дробной части десятичной дроби

На выходе получаются три стека, в которых хранятся символы из выделенных целой, дробной частей и порядка. Для вычисления значения численной константы используется следующий алгоритм:

- Объявить переменные типа числа с плавающей запятой: *M, p,* положить их равными нулю.
- Пока стек цифр целой части не пуст:

    - Прибавить к M число элемент с вершины стека, умноженный на 10^i (i — номер итерации, в начале равен нулю);
    - Удалить элемент с вершины стека
- Пока стек цифр дробной части не пуст:
    - Прибавить к M число элемент с вершины стека, умноженный на 10^{-i} (i — номер итерации, в начале равен нулю);
    - Удалить элемент с вершины стека
- Пока стек цифр порядка не пуст:
    - Прибавить к p число элемент с вершины стека, умноженный на 10^i (i — номер итерации, в начале равен нулю);
    - Удалить элемент с вершины стека
- Умножить p на –1, если на этапе выделения знака порядка был встречен e–. В противном случае ничего не делать.
- Вычислить значение по формуле $M \times 10^{\mathrm{p}}$.

Для перевода выражения из инфиксной записи в постфиксную применяется следующий алгоритм:

- Создать стек для временного размещения токенов.
- Для каждого токена в инфиксной форме
    - Если токен — число или переменная, поместить в постфиксную форму
    - Если токен (: поместить токен в стек
    - Если токен):
        - Пока на вершине стека не (:
            - извлечь из стека токен и поместить его в постфиксную форму
        - Извлечь из стека (
    - Если токен — операция (присваивания, унарная, бинарная) или функция:
        - Пока стек не пуст и выполняется одно из следующих условий: приоритет операции токена с вершины стека

больше приоритета операции токена из инфиксной записи или приоритеты равны и токен из инфиксной записи — операция, вычисляемая слева направо:

- извлечь из стека токен и поместить его в постфиксную форму

- Поместить токен из инфиксной записи в стек

- Перенести все токены из стека в постфиксную форму

На выходе получается массив токенов, записанных в постфиксной форме.

Для вычисления выражения по постфиксной записи применяется следующий алгоритм:

- Создать стек для размещения токенов.
- Для каждого токена в постфиксной форме
  - Если токен — переменная или число: поместить численное значение токена в стек.
  - Если токен — унарный минус:
    - взять число с вершины стека, умножить его на –1, положить обратно в стек
  - Если токен — функция:
    - взять число с вершины стека, применить к нему функцию, положить обратно в стек
  - Если токен — бинарная операция:
    - взять два числа с вершины стека, применить к ним бинарную операцию, положить обратно в стек
  - Если токен — присваивание:
    - взять два токена с вершины стека. Первый взятый токен назовём левым, второй — правым.
    - присвоить правому токену численное значение левого
    - найти (или создать) переменную в массиве переменных с названием, соответствующим строке токена, поместить значение левого токена в массив переменных

- поместить правый токен в стек

4. Результаты экспериментов

Для подтверждения корректности результатов работы программы было написано тестовое приложение на базе фреймворка Google Test, и также было проведено ручное тестирование пользовательского приложения. В ходе проведённого тестирования была подтверждена корректность работы программы.

Заключение

В результате проведения лабораторной работы была достигнута поставленная цель и выполнены поставленные задачи:

- Для представления стека разработан и реализован интерфейс класса TStack, реализованы его основные методы;
- Разработан алгоритм разбора арифметических выражений;
- Разработаны и реализованы классы TPostfix, Tokenizer и Parser, осуществляющие разбор арифметических выражений, перевод и инфиксной формы записи в постфиксную и вычисление по постфиксной форме записи;
- Разработано пользовательское приложение для работы с арифметическими выражениями.

Литература

1. Кормен, Томас Х. и др. Алгоритмы: построение и анализ, 3-е изд. : Пер. с англ. – М. : ООО «И. Д. Вильямс», 2013. – 1328 с. : ил. – Парал. тит. англ.

Приложение

Приложение №1. arithmetic.cpp

```
// реализация функций и классов для вычисления арифметических выражений

#include "arithmetic.h"
#include <sstream>
#include <iostream>

void throw_error(const std::string& s, const std::string& err_s, const size_t pos)
{
	std::stringstream ss;
	ss << err_s
		<< ", position "
		<< pos + 1 << ": "
		<< s.substr(0, pos)
		<< "["
		<< s[pos]
		<< "]"
		<< s.substr(pos + 1, s.size());
	throw std::runtime_error(ss.str());
}

void throw_error(const std::string& s, const std::string& err_s, const std::string& t,
const size_t pos)
{
	std::stringstream ss;
	ss << err_s
		<< ", position "
		<< pos + 1 << "-" << pos + t.size() << ": "
		<< s.substr(0, pos)
		<< "["
		<< t
		<< "]"
		<< s.substr(pos + t.size(), s.size());
	throw std::runtime_error(ss.str());
}

TPostfix::TPostfix()
{
	variables["e"] = 2.71828182845904523536;
	variables["pi"] = 3.14159265358979323846;
}

double TPostfix::evaluate(const std::string& s)
{
	try
	{
		size_t sz = 0;
		Token* tokens = t.tokenize(s, variables, sz);
		Token* postfix = p.convertToPostfix(tokens, sz);
		TStack<Token> tmp;

		for(size_t i = 0; i < sz; i++)
		{
			switch (postfix[i].type)
			{
			case NUM:
			{
				tmp.push(postfix[i]);
				break;
			}

			case VAR:
			{
```

```
                        std::map<std::string, double>::iterator it =
variables.find(postfix[i].s);
                        if (it != variables.end())
                                tmp.push(Token{ it->first, VAR, it->second });
                        else
                                tmp.push(postfix[i]);
                        break;
                }

                case UN_OP:
                {
                        if (postfix[i].s == "-")
                                tmp.top().val *= -1.0;

                        break;
                }

                case BIN_OP:
                {
                        Token lhs = tmp.top(); tmp.pop();
                        Token rhs = tmp.top(); tmp.pop();
                        switch (postfix[i].s[0])
                        {
                        case '+':
                                tmp.push(Token{ "", NUM, rhs.val + lhs.val });
                                break;

                        case '-':
                                tmp.push(Token{ "", NUM, rhs.val - lhs.val });
                                break;

                        case '*':
                                tmp.push(Token{ "", NUM, rhs.val * lhs.val });
                                break;

                        case '/':
                                tmp.push(Token{ "", NUM, rhs.val / lhs.val });
                                break;

                        case '^':
                                tmp.push(Token{ "", NUM, std::pow(rhs.val, lhs.val) });
                                break;
                        }

                        break;
                }

                case FUNC:
                {
                        Token arg = tmp.top(); tmp.pop();

                        if(postfix[i].s == "sin")
                                tmp.push(Token{ "", NUM, std::sin(arg.val) });

                        else if (postfix[i].s == "cos")
                                tmp.push(Token{ "", NUM, std::cos(arg.val) });

                        else if (postfix[i].s == "tan")
                                tmp.push(Token{ "", NUM, std::tan(arg.val) });

                        else if (postfix[i].s == "asin")
                                tmp.push(Token{ "", NUM, std::asin(arg.val) });

                        else if (postfix[i].s == "acos")
                                tmp.push(Token{ "", NUM, std::acos(arg.val) });

                        else if (postfix[i].s == "atan")
```

```
                                    tmp.push(Token{ "", NUM, std::atan(arg.val) });

                              else if (postfix[i].s == "exp")
                                    tmp.push(Token{ "", NUM, std::exp(arg.val) });

                              else if (postfix[i].s == "ln")
                                    tmp.push(Token{ "", NUM, std::log(arg.val) });

                              else if (postfix[i].s == "sqrt")
                                    tmp.push(Token{ "", NUM, std::sqrt(arg.val) });

                              break;
                        }

                        case ASSGN:
                        {
                              Token lhs = tmp.top(); tmp.pop();
                              Token rhs = tmp.top(); tmp.pop();

                              std::map<std::string, double>::iterator it =
variables.find(rhs.s);
                              rhs.val = lhs.val;
                              if (it != variables.end())
                                    it->second = lhs.val;
                              else
                                    variables[rhs.s] = rhs.val;
                              tmp.push(rhs);
                              break;
                        }

                        }
                  }

                  delete[] tokens;
                  delete[] postfix;
                  return tmp.top().val;
            }

            catch (const std::exception& e)
            {
                  throw e;
            }

            return 0.0;
}

const std::map<std::string, double>& TPostfix::getVariables() noexcept
{
      return variables;
}

void TPostfix::clearVariables()
{
      variables.clear();
      variables["e"] = 2.71828182845904523536;
      variables["pi"] = 3.14159265358979323846;
}

TPostfix::Token* TPostfix::Tokenizer::tokenize(const std::string& s,
std::map<std::string, double>& vars, size_t& sz)
{
      TStack<Token> tmp;
      // стек для помещения позиций скобок в данной строке
      TStack<size_t> parenthesis;
      TokenizerState ts = TOKEN_INIT;
      bool unary = true;
      long parenthesisCount = 0;
```

```
	long assignmentCount = 0;
	long expressionLen = 0;
	size_t tokens_count = 0;

	if (s.size() == 0) throw std::runtime_error("An empty string was given");

	for (size_t i = 0; i < s.size();)
	{
		Token t;
		std::string token_string;

		switch (ts)
		{
		case TOKEN_INIT:
			if (s[i] == ' ' || s[i] == '\t' || s[i] == '\r' || s[i] == '\n' ||
s[i] == '\v' || s[i] == '\f') i++;
			else if (s[i] == '.' || (s[i] >= '0' && s[i] <= '9')) ts =
TOKENIZE_NUM;
			else if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/' ||
s[i] == '^') ts = TOKENIZE_OP;
			else if (s[i] == '(') ts = TOKENIZE_LEFT_PAR;
			else if (s[i] == ')') ts = TOKENIZE_RIGHT_PAR;
			else if (s[i] == '=') ts = TOKENIZE_ASSGN;
			else if ((s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z'))
ts = TOKENIZE_WORD;
			else throw_error(s, "Unexpected character", i);
			continue;

		case TOKENIZE_NUM:
			if (!tmp.isEmpty())
			{
				if ((assignmentCount < 1 || tmp.top().type == ASSGN) &&
(tmp.top().type == NUM || tmp.top().type == VAR || tmp.top().type == UNRECG ||
tmp.top().type == RIGHT_PARS))
					throw_error(s, "Unexpected token", i);
			}

			t = tokenizeNumber(s, i);
			tmp.push(t);
			ts = TOKEN_INIT;
			expressionLen++;
			tokens_count++;
			unary = false;
			continue;

		case TOKENIZE_OP:
			t = tokenizeOperation(s, i, unary);

			if (!tmp.isEmpty())
			{
				if (t.type == UN_OP)
				{
					if ((assignmentCount < 1 || tmp.top().type != ASSGN) &&
tmp.top().type != UN_OP && tmp.top().type != BIN_OP && tmp.top().type != FUNC &&
tmp.top().type != LEFT_PARS && tmp.top().type != UNRECG)
						throw_error(s, "Unexpected token", i-1);
				}

				else
				{
					if ((assignmentCount < 1|| tmp.top().type != ASSGN) &&
tmp.top().type != NUM && tmp.top().type != VAR && tmp.top().type != RIGHT_PARS &&
tmp.top().type != UNRECG)
						throw_error(s, "Unexpected token", i-1);
				}
			}
```

```
                    tmp.push(t);
                    ts = TOKEN_INIT;
                    expressionLen++;
                    tokens_count++;
                    unary = true;
                    continue;

             case TOKENIZE_WORD:
                    t = tokenizeWord(s, i);

                    if (vars.count(t.s) > 0)
                    {
                           t.val = vars[t.s];
                           t.type = VAR;
                    }

                    if (!tmp.isEmpty())
                    {
                           if ((assignmentCount < 1 || tmp.top().type != ASSGN) &&
tmp.top().type != UN_OP && tmp.top().type != BIN_OP && tmp.top().type != FUNC &&
tmp.top().type != LEFT_PARS && tmp.top().type != UNRECG)
                                  throw_error(s, "Unexpected token", t.s, i-
t.s.size());
                    }

                    tmp.push(t);
                    ts = TOKEN_INIT;
                    expressionLen++;
                    tokens_count++;
                    t.type == FUNC ? unary = true : unary = false;
                    continue;

             case TOKENIZE_LEFT_PAR:
                    if (!tmp.isEmpty())
                    {
                           if ((assignmentCount < 1 || tmp.top().type != ASSGN) &&
tmp.top().type != LEFT_PARS && tmp.top().type != UN_OP && tmp.top().type != BIN_OP &&
tmp.top().type != FUNC && tmp.top().type != UNRECG)
                                  throw_error(s, "Unexpected token", i);
                    }

                    parenthesis.push(i);
                    t.s = s[i++];
                    t.type = LEFT_PARS;
                    parenthesisCount++;

                    tmp.push(t);
                    ts = TOKEN_INIT;
                    tokens_count++;
                    unary = true;
                    continue;

             case TOKENIZE_RIGHT_PAR:
                    if (parenthesis.isEmpty()) throw_error(s, "Misplaced parenthesis", i);
                    parenthesis.pop();

                    if (!tmp.isEmpty())
                    {
                           if (tmp.top().type != RIGHT_PARS && tmp.top().type != NUM &&
tmp.top().type != VAR && tmp.top().type != UNRECG) throw_error(s, "Unexpected token", i);
                    }

                    t.s = s[i++];
                    t.type = RIGHT_PARS;

                    tmp.push(t);
                    ts = TOKEN_INIT;
```

```
                  tokens_count++;
                  unary = false;
                  continue;

            case TOKENIZE_ASSGN:
                  if (tmp.isEmpty()) throw_error(s, "Unexpected token", i);
                  if (isInvalidVariableName(tmp.top().s)) throw_error(s, "Invalid
variable name", tmp.top().s, i-tmp.top().s.size());
                  if (tmp.top().type != VAR && tmp.top().type != UNRECG) throw_error(s,
"Unexpected token", i);
                  else if (expressionLen > 1) throw_error(s, "Unexpected token", i);
                  if (tmp.top().type == UNRECG) tmp.top().type = VAR;

                  t.s = s[i++];
                  t.type = ASSGN;
                  assignmentCount++;

                  tmp.push(t);
                  ts = TOKEN_INIT;
                  unary = true;
                  expressionLen = 0;
                  tokens_count++;
                  continue;
            }
      }

      if (tokens_count == 0) throw std::runtime_error("An empty string was given");
      if (tmp.top().type == UN_OP || tmp.top().type == BIN_OP || tmp.top().type == FUNC
|| tmp.top().type == ASSGN) throw_error(s, "Missing operand", tmp.top().s, tmp.size()-1);
      if (!parenthesis.isEmpty()) throw_error(s, "Unclosed parenthesis",
parenthesis.top());

      Token* tokens = new Token[tokens_count];
      size_t i = tokens_count - 1;
      size_t pos = s.size() - 1;
      sz = tokens_count;
      while(!tmp.isEmpty())
      {
            while (s[pos] == ' ' || s[pos] == '\t' || s[pos] == '\r' || s[pos] == '\n'
|| s[pos] == '\v' || s[pos] == '\f') pos--;
            pos -= tmp.top().s.size();

            switch (tmp.top().type)
            {
            case UNRECG:
                  throw_error(s, "Unrecognized token", tmp.top().s, pos+1);
                  break;
            default:
                  tokens[i--] = tmp.top();
                  tmp.pop();
                  break;
            }

      }

      return tokens;

}
TPostfix::Token TPostfix::Tokenizer::tokenizeNumber(const std::string& s, size_t& i)
{
      NumberTokenizerState ntst = NT_INIT;
      Token num;

      TStack<char> beforePointChars;
      TStack<char> afterPointChars;
      TStack<char> expChars;
```

```
	size_t first_char = i;
	size_t last_char = i;
	double val = 0.0;
	double pow = 1.0;
	double pow_frac = 0.1;
	double sign = 1.0;
	double exp = 0.0;

	for (; i < s.size();)
	{
		switch (ntst)
		{
		case NT_INIT:
			last_char = i;
			if (s[i] == '.') { i++; ntst = NUM2; }
			else if (s[i] >= '1' && s[i] <= '9') { ntst = NUM1; }
			else if (s[i] == '0') { ntst = ZERO_FIRST; }
			continue;

		case NUM1:
			last_char = i;
			if (s[i] >= '0' && s[i] <= '9') beforePointChars.push(s[i++]);
			else if (s[i] == '.') { i++; ntst = NUM2; }
			else if (s[i] == 'e') { i++; ntst = EXP; }
			else break;
			continue;

		case NUM2:
			last_char = i;
			if (s[i] >= '0' && s[i] <= '9') afterPointChars.push(s[i++]);
			else if (s[i] == 'e') { i++; ntst = EXP; }
			else break;
			continue;

		case NUM3:
			last_char = i;
			if (s[i] >= '0' && s[i] <= '9') expChars.push(s[i++]);
			else if (s[i] == '.') { throw_error(s, "Invalid number format",
last_char); }
			else break;
			continue;

		case EXP:
			last_char = i;
			if (s[i] == '+') i++;
			else if (s[i] == '-') { i++; sign = -1.0; }
			else throw_error(s, "Invalid number format", last_char);

			if (s[i] >= '1' && s[i] <= '9') ntst = NUM3;
			else throw_error(s, "Invalid number format", (i < s.size() ? i : i-
1));
			continue;

		case ZERO_FIRST:
			last_char = i;
			if (s[++i] == '.') { i++; ntst = NUM2; }
			else break;
			continue;
		}

		break;
	}

	if (s[i-1] == '.' && (i - first_char < 2)) throw_error(s, "Invalid number format",
last_char);
	if (ntst != NUM1 && ntst != NUM2 && ntst != NUM3 && ntst != ZERO_FIRST)
		throw_error(s, "Invalid number format", last_char);
```

```
	while (!beforePointChars.isEmpty())
	{
		val += pow * (beforePointChars.top() - '0');
		pow *= 10;
		beforePointChars.pop();
	}

	// перевод в double
	TStack<char> tmp;
	while (!afterPointChars.isEmpty())
	{
		tmp.push(afterPointChars.top());
		afterPointChars.pop();
	}

	while (!tmp.isEmpty())
	{
		val += pow_frac * (tmp.top() - '0');
		pow_frac *= 0.1;
		tmp.pop();
	}

	pow = 1;
	while (!expChars.isEmpty())
	{
		exp += pow * (expChars.top() - '0');
		pow *= 10;
		expChars.pop();
	}

	num.s = s.substr(first_char, i-first_char);
	num.type = NUM;
	num.val = val * std::pow(10, sign * exp);
	return num;
}

TPostfix::Token TPostfix::Tokenizer::tokenizeOperation(const std::string& s, size_t& i,
bool unary)
{
	Token op;
	op.val = std::numeric_limits<double>::quiet_NaN();

	if (unary && s[i] == '-') op.type = UN_OP;
	else if (unary && s[i] != '-') throw_error(s, "Missing operand.", i);
	else op.type = BIN_OP;

	op.s = s[i++];
	return op;
}

TPostfix::Token TPostfix::Tokenizer::tokenizeWord(const std::string& s, size_t& i)
{
	Token word;
	word.val = std::numeric_limits<double>::quiet_NaN();
	std::string t = "";
	size_t first_char = i;

	for (; i < s.size();)
	{
		if (s[i] == ' ' || s[i] == '\t' || s[i] == '\n') break;
		else if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/' || s[i] ==
'=' || s[i] == '^') break;
		else if (s[i] == '(' || s[i] == ')') break;
		t += s[i++];
	}
```

```
	word.s = t;
	if (t == "sin" || t == "cos" || t == "tan" || t == "asin" || t == "acos" || t == "atan" || t == "ln" || t == "exp" || t == "sqrt")
		word.type = FUNC;
	else word.type = UNRECG;

	return word;
}

inline bool TPostfix::Tokenizer::isInvalidVariableName(const std::string& s) noexcept
{
	if (s == "sin" || s == "cos" || s == "tan" || s == "asin" || s == "acos" || s == "atan" || s == "ln" || s == "exp" || s == "sqrt" ||
		s == "" || s == " " || s == "\t" || s == "\n" ||
		s == "+" || s == "-" || s == "*" || s == "/" || s == "^" || s == "(" || s == ")" || s == "=" || s == "e" || s == "pi" || s == "help" || s == "quit" || s == "vars" || s == "clear") return true;
	else
	{
		for (size_t i = 0; i < s.size(); i++)
			if ( !( (s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z') || (s[i] >= '0' && s[i] <= '9') ) ) return true;
	}

	return false;
}

TPostfix::Token* TPostfix::Parser::convertToPostfix(Token* tokens, size_t& sz)
{
	TStack<Token> postfix;
	TStack<Token> tmp;
	size_t inp_sz = sz;

	for(size_t i = 0; i < inp_sz; i++)
	{
		switch (tokens[i].type)
		{
		case NUM: {}
		case VAR:
		{
			postfix.push(tokens[i]);
			break;
		}

		case LEFT_PARS:
		{
			tmp.push(tokens[i]);
			sz--;
			break;
		}

		case RIGHT_PARS:
		{
			while (tmp.top().type != LEFT_PARS)
			{
				postfix.push(tmp.top());
				tmp.pop();
			}

			tmp.pop();
			sz--;
			break;
		}

		case UN_OP: {}
		case ASSGN: {}
		case FUNC: {}
```

```
			case BIN_OP:
			{
				while((!tmp.isEmpty()) && (operatorPriority(tmp.top()) >
operatorPriority(tokens[i])
					|| (operatorPriority(tmp.top()) == operatorPriority(tokens[i])
&& isLeftAssoc(tokens[i]))))
				{
					postfix.push(tmp.top());
					tmp.pop();
				}

				tmp.push(tokens[i]);
				break;
			}

			default:
				throw std::runtime_error("Unknown error");
				break;
			}
		}

		while (!tmp.isEmpty())
		{
			postfix.push(tmp.top());
			tmp.pop();
		}

		Token* res = new Token[sz];
		size_t j = sz - 1;
		while (!postfix.isEmpty())
		{
			res[j--] = postfix.top();
			postfix.pop();
		}

		return res;
}

inline int TPostfix::Parser::operatorPriority(const Token& t) noexcept
{
		if (t.type == BIN_OP)
		{
			if (t.s == "+" || t.s == "-") return 2;
			else if (t.s == "*" || t.s == "/") return 3;
			else if (t.s == "^") return 4;
		}

		else if (t.type == UN_OP) return 4;
		else if (t.type == FUNC) return 5;
		else if (t.type == ASSGN) return 1;
		return -1;
}

inline bool TPostfix::Parser::isLeftAssoc(const Token& t) noexcept
{
		if (t.type == UN_OP || t.type == FUNC || t.type == ASSGN) return false;
		else if (t.s == "^") return false;
		return true;

}
```

Приложение №2. stack.h

```
// объявление и реализация шаблонного стека
// стек поддерживает операции:
// - вставка элемента,
// - извлечение элемента,
// - просмотр верхнего элемента (без удаления)
// - проверка на пустоту,
// - получение количества элементов в стеке
// - очистка стека
// при вставке в полный стек должна перевыделяться память

template <class T>
class TStack
{
public:
	TStack() : sz(0), cap(256) { data = new T[cap]; }
	TStack(const TStack& s) : sz(s.sz), cap(s.cap)
	{
		data = new T[cap];
		std::copy(s.data, s.data + sz, data);
	}

	TStack(TStack&& s) noexcept
	{
		sz = 0;
		cap = 0;
		data = nullptr;
		swap(*this, s);
	}

	~TStack()
	{
		sz = 0;
		cap = 0;
		delete[] data;
		data = nullptr;
	}

	TStack& operator=(const TStack& s)
	{
		if (this == &s)
			return *this;
		TStack tmp(s);
		swap(*this, tmp);
		return *this;
	}

	TStack& operator=(TStack&& s) noexcept
	{
		delete[] data;
		sz = 0;
		cap = 0;
		data = nullptr;
		swap(*this, s);
		return *this;
	}

	friend void swap(TStack& lhs, TStack& rhs) noexcept
	{
		std::swap(lhs.sz, rhs.sz);
		std::swap(lhs.cap, rhs.cap);
		std::swap(lhs.data, rhs.data);
	}

	void push(const T& value)
	{
		if (sz == cap) resize();
```

```
            data[sz++] = value;
        }

        void push(T&& value)
        {
            if (sz == cap) resize();
            data[sz++] = std::move(value);
        }

        void pop()
        {
            if (!isEmpty()) { sz--; }
            else throw std::runtime_error("Trying to pop from empty stack");
        }

        void clear()
        {
            TStack s;
            swap(*this, s);
        }

        T& top()
        {
            if (sz == 0) throw std::runtime_error("Trying to get element from empty
stack");
            return data[sz-1];
        }

        const T& top() const
        {
            if (sz == 0) throw std::runtime_error("Trying to get element from empty
stack");
            return data[sz-1];
        }

        bool isEmpty() noexcept { return sz == 0; }
        size_t size() noexcept { return sz; }

    private:
        T* data;
        size_t sz;
        size_t cap;

        void resize()
        {
            T* tmp = new T[cap * 2];
            std::copy(data, data + sz, tmp);
            cap *= 2;
            delete[] data;
            data = tmp;
        }

    };
```