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

Леонтьев Н.С.

**Проверил**:

преподаватель каф. МОСТ, Волокитин В.Д.

Нижний Новгород

2023

Содержание

[Введение 3](#_Toc529541653)

[1. Постановка задачи 4](#_Toc529541654)

[2. Руководство пользователя 5](#_Toc529541655)

[3.1. Описание структуры программы 6](#_Toc529541656)

[3.2. Описание алгоритмов 6](#_Toc529541657)

[4. Результаты экспериментов 8](#_Toc529541658)

[Заключение 10](#_Toc529541659)

[Литература 11](#_Toc529541660)

[Приложение 12](#_Toc529541661)

# Введение

Нередко возникает потребность в обработке арифметических выражений, вместо выполнения одной операции за раз. Она часто возникает при разработке калькуляторов, построителей графиков и т.д.

Итог для пользователя: возможность использовать переменные, численные значения для вычисления арифметического выражения.

По этой причине тему лабораторной работы можно считать актуальной.

# Постановка задачи

Цель данной работы — разработка структуры данных Стек и ее использование для расчета арифметических выражений с использованием обратной польской записи (постфиксной формы).

1. Выполнение работы предполагает решение следующих задач:
2. Разработка интерфейса шаблонного класса TStack.
3. Реализация методов шаблонного класса TStack.
4. Разработка интерфейса класса TPostfix для работы с постфиксной формой.
5. Реализация методов класса TPostfix.
6. Разработка и реализация тестов для классов TStack и TPostfix на базе Google Test.
7. Публикация исходных кодов в личном репозитории на GitHub.

# Руководство пользователя

Для открытия проекта его нужно либо собрать через CMake, либо использовать Visual Studio проект в директории sln.

Для начала программа предложит (рис. 1) проверить ввод чисел с плавающей точкой для примера, как нужно их вводить.

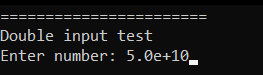


рис. 1. «Введите число»

После введения числа программа бесконечно будет просить ввести арифметическое выражение (рис. 2).

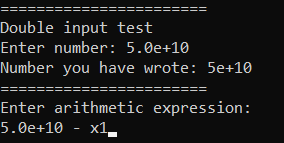


рис. 2. «Введите арифметическое выражение»

После ввода программа спросит значения переменных, и при корректности ввода выведет значение выражения. Иначе выдаст ошибки (рис. 3).

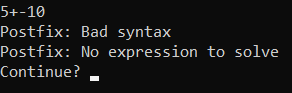


рис. 3. «Плохой синтаксис»

При необходимости закрыть программу пользователь может ввести ‘n’ либо ‘N’.

1. **Руководство программиста**

## Описание структуры программы

**gtest** — библиотека Google Test.

**samples** — каталог с пользовательским приложением.

**test** — каталог с проектом с модульными тестами.

**include, src** - каталоги с основными файлами: arithmetic.h, arithmetic.cpp, stack.h.

**sln** - каталог с файлами с решениями (solution) для Microsoft Visual Studio 2010 и 2012.

**README.md** — информация о проекте.

Служебные файлы:

**.gitignore** — перечень расширений файлов, игнорируемых Git при добавлении файлов в репозиторий.

## Описание алгоритмов

Библиотека stack.h позволяет работать со структурой данных Stack (неограниченный стек) через шаблонный класс TStack. Это его основная задача.

В нём определён конструктор по умолчанию, конструктор копирования, оператор =. Методы класса TStack:

void push(T elem); // добавить элемент в стек

T top(); // посмотреть элемент

T pop(); // вытащить элемент

void clear(); // очистить стек

size\_t get\_size(); // получить кол-во элементов

bool isFull(); // полный относительно выделенной памяти?

bool isEmpty(); // пустой?

Библиотека arithmetic.h предоставляет программисту методы для работы с арифметическими выражениями, а также класс TPostfix.

Доступные методы:

double calculate\_expr(const string &expr, istream& values\_stream = std::cin); // значение выражения

1) expr: выражение;

2) values\_stream: поток, куда будут входить переменные.

double string\_to\_decimal(const string &expr); // число в текстовом формате в дробную часть нуля (0123 -> 0.0123).

int string\_to\_integer(const string &expr); // число в текстовом формате в целую часть числа (0123 -> 123).

double exp\_form\_to\_double(const string &expr); // экспоненциальная форма записи числа в число с плавающей точкой.

size\_t operator\_priority(const string &expr); // приоритеты имеющихся операций.

inline bool is\_arithmetical(char inp); // проверка является ли символ операцией (+-\*/) или скобками.

inline bool is\_alphabet\_or\_numeric(char inp); // проверка является ли символом латинского алфавита, цифрой или точкой.

vector<string> arithmetic\_to\_lexems(const string &expr); // разделить выражение expr на лексемы.

bool check\_infix\_correctness(const vector<string>& lexems); // проверка корректности арифметического выражения.

Шаблонный класс TPostfix содержит следующие методы:

void toPostfix(const vector<string> &expr); // лексемы в постфикс.

T calculate(); // рассчитать и вернуть значение

void askOperands(istream& from, ostream& log = cout); // спросить у пользователя переменные, используя два потока.

Функции exp\_form\_to\_double и check\_infix\_correctness были реализованы с использованием конечных автоматов, а точнее матриц, позволяющих перемещаться по конечному автомату.

# Результаты экспериментов

Результаты экспериментов представлены в тестах. Код проходит все из них.

[==========] Running 23 tests from 2 test cases.

[----------] Global test environment set-up.

[----------] 5 tests from TPostfix

[ RUN ] TPostfix.can\_do\_simple\_operations

[ OK ] TPostfix.can\_do\_simple\_operations (0 ms)

[ RUN ] TPostfix.can\_do\_complex\_operations

[ OK ] TPostfix.can\_do\_complex\_operations (1 ms)

[ RUN ] TPostfix.throws\_when\_incorrect\_function\_usage

[ OK ] TPostfix.throws\_when\_incorrect\_function\_usage (2 ms)

[ RUN ] TPostfix.common\_mistakes

[ OK ] TPostfix.common\_mistakes (4 ms)

[ RUN ] TPostfix.input\_numbers

[ OK ] TPostfix.input\_numbers (0 ms)

[----------] 5 tests from TPostfix (8 ms total)

[----------] 18 tests from TStack

[ RUN ] TStack.can\_create\_stack

[ OK ] TStack.can\_create\_stack (0 ms)

[ RUN ] TStack.can\_copy\_stack

[ OK ] TStack.can\_copy\_stack (0 ms)

[ RUN ] TStack.copied\_class\_has\_own\_memory

[ OK ] TStack.copied\_class\_has\_own\_memory (0 ms)

[ RUN ] TStack.can\_assign\_stack

[ OK ] TStack.can\_assign\_stack (0 ms)

[ RUN ] TStack.can\_do\_self\_assignment

[ OK ] TStack.can\_do\_self\_assignment (0 ms)

[ RUN ] TStack.assignment\_operator\_assigns\_size

[ OK ] TStack.assignment\_operator\_assigns\_size (0 ms)

[ RUN ] TStack.can\_push\_element

[ OK ] TStack.can\_push\_element (0 ms)

[ RUN ] TStack.can\_pop\_element

[ OK ] TStack.can\_pop\_element (0 ms)

[ RUN ] TStack.cant\_pop\_if\_empty\_stack

[ OK ] TStack.cant\_pop\_if\_empty\_stack (1 ms)

[ RUN ] TStack.can\_get\_top\_element

[ OK ] TStack.can\_get\_top\_element (0 ms)

[ RUN ] TStack.top\_element\_changes\_on\_push

[ OK ] TStack.top\_element\_changes\_on\_push (0 ms)

[ RUN ] TStack.top\_element\_changes\_on\_pop

[ OK ] TStack.top\_element\_changes\_on\_pop (0 ms)

[ RUN ] TStack.cant\_get\_top\_if\_empty\_stack

[ OK ] TStack.cant\_get\_top\_if\_empty\_stack (1 ms)

[ RUN ] TStack.stack\_size\_returns\_zero\_if\_empty\_stack

[ OK ] TStack.stack\_size\_returns\_zero\_if\_empty\_stack (0 ms)

[ RUN ] TStack.stack\_size\_increases\_on\_push

[ OK ] TStack.stack\_size\_increases\_on\_push (0 ms)

[ RUN ] TStack.stack\_size\_decreases\_on\_pop

[ OK ] TStack.stack\_size\_decreases\_on\_pop (0 ms)

[ RUN ] TStack.isempty\_returns\_true\_if\_empty

[ OK ] TStack.isempty\_returns\_true\_if\_empty (0 ms)

[ RUN ] TStack.isempty\_returns\_false\_if\_not\_empty

[ OK ] TStack.isempty\_returns\_false\_if\_not\_empty (0 ms)

[----------] 18 tests from TStack (44 ms total)

[----------] Global test environment tear-down

[==========] 23 tests from 2 test cases ran. (53 ms total)

[ PASSED ] 23 tests.

Код тестов представлен в директории test

# Заключение

Была разработана библиотека, работающая с арифметическими выражениями, а также со структурой данных Stack. Цели и задачи лабораторной работы считаю выполненными.

# Литература

1. Лекции Александра Владимировича Сысоева.

# Приложение

double calculate\_expr(const string &expr, istream& values\_stream) {

TPostfix<double> postfix;

vector<string> lexems = arithmetic\_to\_lexems(expr);

postfix.toPostfix(lexems);

postfix.askOperands(values\_stream);

double res;

res = postfix.calculate();

return res;

}

double string\_to\_decimal(const string &expr) {

double res = 0.0;

if (expr.size() == 0) {

return res;

}

for (size\_t i = expr.size() - 1; i > 0; i--) {

res += static\_cast<double>(expr[i]) - static\_cast<double>('0');

res /= 10;

}

if (expr[0] == '-') {

res \*= -1;

}

else {

res += static\_cast<double>(expr[0]) - static\_cast<double>('0');

res /= 10;

}

return res;

}

int string\_to\_integer(const string &expr) {

int res = 0;

bool minus = false;

if (expr.size() == 0) {

return res;

}

size\_t i = 0;

if (expr[0] == '-') {

minus = true;

i++;

}

while (i < expr.size()) {

res \*= 10;

res += static\_cast<int>(expr[i]) - static\_cast<int>('0');

i++;

}

if (minus) {

res \*= -1;

}

return res;

}

double exp\_form\_to\_double(const string &expr) {

// init state machine matrix

const size\_t a\_size = 6;

const size\_t s\_size = 10;

const size\_t a0[a\_size] = { 0, 0, 0, 0, 0, 0 };

const size\_t a1[a\_size] = { 4, 3, 2, 6, 0, 0 };

const size\_t a2[a\_size] = { 4, 3, 0, 6, 0, 0 };

const size\_t a3[a\_size] = { 3, 3, 0, 5, 7, 0 };

const size\_t a4[a\_size] = { 0, 0, 0, 5, 7, 0 };

const size\_t a5[a\_size] = { 5, 5, 0, 0, 7, 0 };

const size\_t a6[a\_size] = { 5, 5, 0, 0, 0, 0 };

const size\_t a7[a\_size] = { 0, 0, 8, 0, 0, 8 };

const size\_t a8[a\_size] = { 9, 9, 0, 0, 0, 0 };

const size\_t a9[a\_size] = { 9, 9, 0, 0, 0, 0 };

const size\_t \*states\_array[s\_size] = { a0, a1, a2, a3, a4, a5, a6, a7, a8, a9 };

// get 3 nums

string first\_num = "";

string mov = "";

string second\_num = "";

size\_t cur\_state = 1;

size\_t cur\_slot = 0;

char item;

for (size\_t i = 0; i < expr.size(); i++) {

// move state

item = expr[i];

if (item == '0') { cur\_state = states\_array[cur\_state][0]; }

else if (item >= '1' && item <= '9') { cur\_state = states\_array[cur\_state][1]; }

else if (item == '-') { cur\_state = states\_array[cur\_state][2]; }

else if (item == '.') { cur\_state = states\_array[cur\_state][3]; }

else if (item == 'e') { cur\_state = states\_array[cur\_state][4]; }

else if (item == '+') { cur\_state = states\_array[cur\_state][5]; }

else {

//throw std::invalid\_argument("Bad number");

continue;

}

if (cur\_state == 0) {

break;

}

if (item == '.') {

cur\_slot = 1;

}

else if (item == 'e') {

cur\_slot = 2;

}

else if (item == '-' || (item >= '0' && item <= '9')) {

// add to corresponding num

switch (cur\_slot) {

case 0:

first\_num += item;

break;

case 1:

second\_num += item;

break;

case 2:

mov += item;

break;

}

}

}

// find result

double res;

res = string\_to\_integer(first\_num) + string\_to\_decimal(second\_num);

res \*= std::pow(10, string\_to\_integer(mov));

return res;

}

size\_t operator\_priority(const string &expr) {

if (expr == "+") { return 1; }

else if (expr == "-") { return 1; }

else if (expr == "\*") { return 2; }

else if (expr == "/") { return 2; }

else if (expr == "~") { return 2; }

else if (expr == "sin") { return 3; }

else if (expr == "cos") { return 3; }

else if (expr == "tan") { return 3; }

else if (expr == "exp") { return 3; }

else if (expr == "lg") { return 3; }

else { return 0; }

}

vector<string> arithmetic\_to\_lexems(const string &expr) {

vector<string> res;

string buf = "";

char item;

// when +,- is part of number

bool expcase;

for (size\_t i = 0; i < expr.size(); i++) {

item = expr[i];

expcase = ((i > 0) && (item == '+' || item == '-') && (expr[i - 1] == 'e'));

if (is\_arithmetical(item) && !expcase) {

if (buf != "") {

res.push\_back(buf);

buf = "";

}

if (item == '-' && ((i == 0) || (expr[i-1] == '('))) {

buf += '~';

}

else {

buf += item;

}

res.push\_back(buf);

buf = "";

}

else if (is\_alphabet\_or\_numeric(item) || expcase) {

buf += item;

}

}

if (buf != "") {

res.push\_back(buf);

}

return res;

}

bool check\_infix\_correctness(const vector<string> &lexems) {

// init state machine matrix

const size\_t a\_size = 6;

const size\_t s\_size = 7;

const size\_t a0[a\_size] = { 0, 0, 0, 0, 0, 0 };

const size\_t a1[a\_size] = { 0, 1, 2, 4, 0, 3 };

const size\_t a2[a\_size] = { 6, 0, 0, 0, 5, 0 };

const size\_t a3[a\_size] = { 0, 1, 2, 4, 0, 0 };

const size\_t a4[a\_size] = { 0, 1, 0, 0, 0, 0 };

const size\_t a5[a\_size] = { 6, 0, 0, 0, 5, 0 };

const size\_t a6[a\_size] = { 0, 1, 2, 4, 0, 0 };

const size\_t \*states\_array[s\_size] = { a0, a1, a2, a3, a4, a5, a6 };

// check

string lexem;

size\_t cur\_state = 1;

for (size\_t i = 0; i < lexems.size(); i++) {

// move state

lexem = lexems[i];

if ((operator\_priority(lexem) == 1 || operator\_priority(lexem) == 2) && lexem != "~") { cur\_state = states\_array[cur\_state][0]; }

else if (lexem == "(") { cur\_state = states\_array[cur\_state][1]; }

else if (operator\_priority(lexem) == 3) { cur\_state = states\_array[cur\_state][3]; }

else if (lexem == ")") { cur\_state = states\_array[cur\_state][4]; }

else if (lexem == "~") { cur\_state = states\_array[cur\_state][5]; }

else { cur\_state = states\_array[cur\_state][2]; }

if (cur\_state == 0) {

return false;

}

}

if (cur\_state == 2 || cur\_state == 5) {

return true;

}

else {

return false;

}

}