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**

**«Моделирование работы сортировщика файлов ОС»**

**Выполнил**:

студент группы 3822Б1ПМ1

Коротин Е.В.

**Проверил**:

доцент кафедры МОСТ, к.т.н.,

Сысоев А.В.

Нижний Новгород

2018

Содержание

[Введение 3](#_Toc153834221)

[1. Постановка задачи 4](#_Toc153834222)

[2. Руководство пользователя 5](#_Toc153834223)

[3.1. Описание структуры программы 7](#_Toc153834224)

[3.2. Описание алгоритмов 7](#_Toc153834225)

[Заключение 10](#_Toc153834226)

[Приложение 11](#_Toc153834227)

# Введение

В этой лабораторной работе мы разработаем программу для подсчёта арифметических выражений. Для более удобной реализации данной задачи, мы будем записывать наше арифметическое выражение в виде обратной польской записи (ОПЗ). В данной реализации мы используем стек для более удобного перевода выражения в ОПЗ. Таким образом в результате данной лабораторной работы мы должны получить 2 класса (класс для работы с постфиксными выражениями и класс - стек), а также программу, которая будет подсчитывать арифметические выражения, используя эти классы.

# Постановка задачи

1.Разработка интерфейса шаблонного класса TStack.

2.Реализация методов шаблонного класса TStack.

3.Разработка интерфейса класса TPostfix для работы с постфиксной формой.

4.Реализация методов класса TPostfix.

5.Разработка и реализация тестов для классов TStack и TPostfix на базе Google Test.

6. Разработка программы для обработки арифметических выражений с использованием класс TPostfix.

6.Публикация исходных кодов в личном репозитории на GitHub.

# Руководство пользователя

При запуске приложения пользователю будет выведено меню (Рис.1) для ввода арифметического выражения, в котором доступны к использованию:

все арифметические операции (+, -, \*, /),

функции: sin, cos, tan, cot, exp, log (требуется выделять операнды для них с помощью (), например «log(6)»),

любое количество переменных вида x<любой набор чисел> ,

любое количество пробелов в строке (они будут игнорироваться программой, например запись «s in (1. 7)» будет воспринята, как «sin(1.7)»);

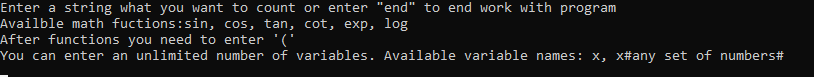
так же пользователю предлагается завершить работу программы, при введении в строку «end».

Рисунок 1

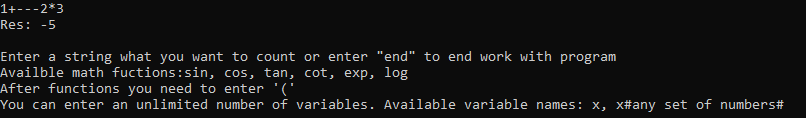
При введении выражения, не содержащего ошибок и переменных, программа выведет его в 1й строке, оно будет посчитано и пользователю будет выведен результат во второй строке, также пользователю будет заново предложено ввести новое выражение или завершить работу программы (Пример такого случая изображён на рисунке 2).

Рисунок 2

****При вводе приложения, содержащего переменные и не содержащего ошибок, программы выведет его в первой строке и предложит пользователю ввести значения всех переменных, которые он ввёл (Пример такого случая изображён на рисунке 3).

Рисунок 3

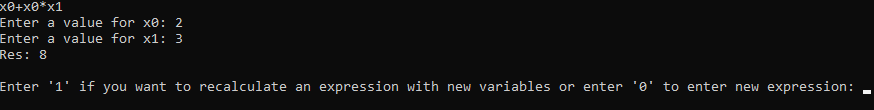
После ввода всех переменных в следующей строке пользователю будет выведен результат этого выражения, посчитанного при выбранных значениях переменных. А также пользователю будет предложено 2 варианта: ввести ‘1’, чтобы пересчитать данное выражение с другими значениями переменных или ввести ‘0’, чтобы ввести другое выражение (Рис. 4).

Рисунок 4

При введении значения, отличного от ‘1’ или ‘0', пользователю будет выведено следующее сообщение см. Рис. 5, а также будет предложено ввести значение его заново.

При введении арифметического выражения с ошибками, оно так же будет выведено в 1й строке, однако после этого будет выведено сообщение-ошибка, указывающее первый символ, который был введён пользователем неверно пользователем неверно, после чего будет выведено сообщение с пояснением произошедшей ошибки. Пользователю будет предложено ввести новое арифметическое выражение (Пример такого случая изображён на рисунке 6).

Рисунок 5

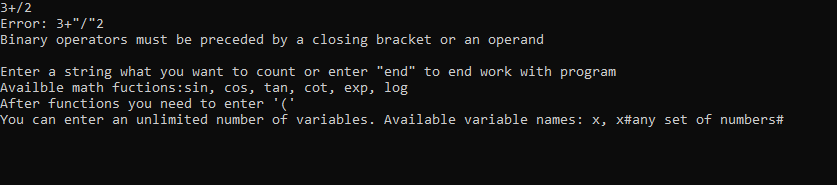
****

Рисунок 6

1. **Руководство программиста**

## Описание структуры программы

Программа состоит из проектов:

arithmetic – содержащего заголовочные файлы шаблонного класса TStack и класса TPostfix (stack.h и arithmetic.h), а также файл arithmetic.cpp с реализацией класса TPostfix;

gtest – библиотека Google Test;

sample – содержащего само приложение для обработки арифметических выражений:

tests – содержащего файлы test\_arithmetic.cpp, test\_stack.cpp и test\_main.cpp, в которых находятся тест, проверяющие корректность работы классов TStack и TPostfix.

## Описание алгоритмов

Шаблонный класс TStack реализует стек, поддерживающий операции:

вставка элемента (void push\_back(const T& val));

извлечение элемента (T pop\_back());

просмотр верхнего элемента стека без удаления (T& top());

проверка стека на пустоту (bool isEmpty() noexcept);

получение количества элементов в стеке (size\_t GetCount() noexcept);

очистка стека (void clear());

private метод bool isFull() noexcept проверяет стек на заполненность;

также с помощью private метода void resize() реализовано перевыделение памяти при вставке элемента в полный стек;

Помимо вышеописанных операций, класс TStack содержит конструктор и деструктор.

Класс TPostfix реализовывает работу с арифметическими выражениями, ниже приведены поле и методы, которые содержит этот класс с кратким их описанием.

std::vector<std::string> RPN Вектор, содержащий строку в виде обратной польской записи.

std::string Error\_string(const std::string& s, int i) Метод, принимающий на вход обрабатываемую строку и индекс символа в этой строке. Метод создаёт новую строку на основе переданной, выделяя в строке элемент под номером “i”. Используется только для выделения неверного элемента во время выбрасывания исключений.

int get\_prior(const std::string& s) noexcept Метод, который возвращает приоритет операции в арифметическом выражении (унарный минус и функции обладают наивысшим приоритетом, \* и / - приоритетнее, чем + и -), передан элемент, не являющийся операцией или функцией, возвращает 0.

int get\_prior(char c) noexcept Такой же метод для символов (не сравнивает переданное значение с функциями).

double valid(const std::string& s) Метод, получающий на вход строку, содержащую проверенное на корректность ввода число, возвращает число, которое было в строке.

std::string number\_check(const std::string& s, int& i) Метод, принимающий обрабатываемую строку и индекс элемента с которого начинается предполагаемое число и проверяет его на корректность, возвращает это число в виде строки.

public:

TPostfix(const std::string& s) Конструктор класса TPostfix , который принимает на вход строку (арифметическое выражение), удаляет все присутствующие в ней пробелы, проверяет её на корректность введения и сохраняет её внутри класса в виде обратной польской записи. В случае, если арифметическое выражение было введено некорректно, выбрасывается исключение, в котором указан первый неверно введённый символ и пояснение произошедшей ошибки.

double count() Считает арифметическое выражение, сохранённое в элементе класса. Переменные вводит пользователь.

double count(double\* variables, int number\_of\_variables) Считает арифметическое выражение, сохранённое в элементе класса. Переменные вводятся внутри программы с помощью указателя на массив и количества переменных (если количество переменных, переданное в этот метод меньше, чем необходимо для подсчёта арифметического выражения, метод выбрасывает исключение).

};

Реализацию данных методов можно найти в приложении к отчёту.

# Заключение

В ходе работы был разработан интерфейс шаблонного класса TStack, а также реализованы все требуемые для него методы. Разработан интерфейс класса TPostfix для работы с постфиксной формой, а также реализованы все методы этого класса.

Разработаны и реализованы тесты для классов TStack и TPostfix на базе Google Test.

Разработана программа для обработки арифметических выражений с использованием класса TPostfix.

Исходные коды были опубликованы в личном репозитории на GitHub.

# Приложение

std::string TPostfix::Error\_string(const std::string& s, int i) {

std::string tmp = "Error: ";

if (i != s.size()) {

for (int j = 0; j < i; j++) tmp += s[j];

tmp = tmp + '\"' + s[i] + '\"';

for (int j = i + 1; j < s.size(); j++) tmp += s[j];

tmp += '\n';

}

else tmp = tmp + s + "\" \"\n";

return tmp;

}

int TPostfix::get\_prior(const std::string& s) noexcept {

if (s == "+" || s == "-") return 1;

if (s == "\*" || s == "/") return 2;

if (s == "~" || s == "sin" || s == "cos" || s == "tan" || s == "cot" || s == "exp" || s == "log") return 3;

return 0;

}

int TPostfix::get\_prior(char c) noexcept {

if (c == '+' || c == '-') return 1;

if (c == '\*' || c == '/') return 2;

if (c == '~') return 3;

return 0;

}

double TPostfix::valid(const std::string& s) {

double res = 0;

int i = 0, mod = 0, pw = 0;

while ((i != s.size()) && (s[i] != '.') && (s[i] != 'e')) {

res \*= 10;

res += (int)s[i++] - 48;

}

int m = i;

if (s[i] == '.') i++;

while ((s[i] != 'e') && (i != s.size())) {

res += ((int)s[i] - 48) \* pow(10, m - i);

i++;

}

if (s[i] == 'e') {

i++;

if (s[i] == '-') mod = -1;

else mod = 1;

i++;

};

while (i != s.size()) {

pw \*= 10;

pw += (int)s[i++] - 48;

}

res \*= pow(10, mod \* pw);

return res;

}

std::string TPostfix::number\_check(const std::string& s, int& i) {

std::string tmp;

tmp = s[i++];

while (i != s.size() && (s[i] >= '0' && s[i] <= '9')) tmp += s[i++];

if (s[i] == '.') tmp += s[i++];

while (i != s.size() && s[i] >= '0' && s[i] <= '9') tmp += s[i++];

if (tmp == ".") throw std::invalid\_argument(Error\_string(s, i - 1) + "Writing a number in the form of \".\" is not possible");

if (i != s.size() && s[i] == 'e') {

tmp += s[i++];

if ((i < s.size() - 1) && (s[i] == '+' || s[i] == '-')) {

tmp += s[i++];

if (s[i] < '1' || s[i]>'9')

throw std::invalid\_argument(Error\_string(s, i) + "When specifying a number with \'e\', there must be a natural number after the \'+\' or \'-\' sign");

tmp += s[i++];

}

else throw std::invalid\_argument(Error\_string(s, i) + "After \'e\' there should be a \'+\' or \'-\'");

};

while (i != s.size() && (s[i] >= '0' && s[i] <= '9')) tmp += s[i++];

if (i != s.size() && !get\_prior(s[i]) && s[i] != ')') throw std::invalid\_argument(Error\_string(s, i) + "The operand must be followed by an operator or a closing parenthesis");

i--;

return tmp;

}

TPostfix::TPostfix(const std::string& str) {

int i = 0;

std::string tmp, s;

for (int j = 0; j < str.size(); j++) if (str[j] != ' ') s += str[j];

if (s.size() == 0) throw std::invalid\_argument("Nothing can be done with an empty string");

TStack<std::string> St;

TStack<int> bracket; //This stack is needed to indicate the first incorrectly plcaed bracket

while (i < s.size()) {

tmp = s[i];

if (get\_prior(s[i])) { //Work with operators

if ( i == 0 || s[i - 1] == '(' || get\_prior(s[i - 1])) {

if (tmp == "-") tmp = "~";

else throw std::invalid\_argument(Error\_string(s, i) + "Binary operators must be preceded by a closing bracket or an operand");

}

if (i == s.size() - 1 || s[i + 1] == ')') throw std::invalid\_argument(Error\_string(s, i+1) + "The operator must be followed by an operand or unary minus");

while (!St.isEmpty() && St.top() != "(" && get\_prior(tmp) != 3 && get\_prior(St.top()) >= get\_prior(tmp)) RPN.push\_back(St.pop\_back());

St.push\_back(tmp);

}

else if (s[i] == '(' && i < s.size() - 1 && s[i + 1] != ')') { //Work with opening bracket

bracket.push\_back(i);

St.push\_back(tmp);

}

else if (s[i] == ')') { //Work with closing bracket

if (bracket.GetCount() == 0) throw std::invalid\_argument(Error\_string(s, i) + "The brackets are placed incorrectly");

bracket.pop\_back();

if (i != s.size() - 1 && !get\_prior(s[i + 1]) && s[i + 1] != ')') throw std::invalid\_argument(Error\_string(s, i + 1) + "The operand must be followed by an operator or a closing bracket");

while (St.top() != "(") RPN.push\_back(St.pop\_back());

St.pop\_back();

}

else if (i < s.size() - 2 && s.size()>2 && get\_prior(tmp + s[i + 1] + s[i + 2])) { //Work with functions

if (i > s.size() - 4 || s.size() < 4 || s[i + 3] != '(') throw std::invalid\_argument(Error\_string(s, i + 3) + "After a function must be a \'(\'");

tmp = tmp + s[i + 1] + s[i + 2];

i += 2;

St.push\_back(tmp);

}

else if ((s[i] >= '0' && s[i] <= '9') || s[i] == '.') { //Work with numbers

if (s[i] == '0'&& s[i+1] >= '0' && s[i+1] <= '9') throw std::invalid\_argument(Error\_string(s, i) + "A number cannot start with an insignificant \'0\'");

RPN.push\_back(number\_check(s, i));

}

else if (s[i] == 'x') { //Work with variables

i++;

while ((i != s.size()) && (s[i] >= '0' && s[i] <= '9')) {

tmp += s[i];

i++;

}

if (i != s.size() && !get\_prior(s[i]) && s[i] != ')') throw std::invalid\_argument(Error\_string(s, i) + "You can't name variables this way");

i--;

RPN.push\_back(tmp);

}

else throw std::invalid\_argument(Error\_string(s, i) + "This writing is not correct");

i++;

}

if (bracket.GetCount() != 0) {

if (bracket.GetCount() > 1) while (bracket.GetCount() != 1) bracket.pop\_back();

throw std::invalid\_argument(Error\_string(s, bracket.pop\_back()) + "The brackets are placed incorrectly");

}

while (!St.isEmpty()) RPN.push\_back(St.pop\_back());

}

double TPostfix::count() {

TStack<double> St;

std::map <std::string, double> variables;

for (int i = 0; i < RPN.size(); i++) {

if (get\_prior(RPN[i])) {

if (RPN[i] == "+") St.push\_back(St.pop\_back() + St.pop\_back());

else if (RPN[i] == "-") St.push\_back(-St.pop\_back() + St.pop\_back());

else if (RPN[i] == "\*") St.push\_back(St.pop\_back() \* St.pop\_back());

else if (RPN[i] == "/") St.push\_back(1 / St.pop\_back() \* St.pop\_back());

else if (RPN[i] == "~") St.top() \*= -1;

else if (RPN[i] == "sin") St.push\_back(sin(St.pop\_back()));

else if (RPN[i] == "cos") St.push\_back(cos(St.pop\_back()));

else if (RPN[i] == "tan") St.push\_back(tan(St.pop\_back()));

else if (RPN[i] == "cot") St.push\_back(1 / tan(St.pop\_back()));

else if (RPN[i] == "exp") St.push\_back(exp(St.pop\_back()));

else St.push\_back(log(St.pop\_back()));

}

else {

if (RPN[i][0] == 'x') {

if (variables.count(RPN[i]))

St.push\_back(variables.find(RPN[i])->second);

else {

double tmp;

std::cout << "Enter a value for " << RPN[i] << ": ";

std::cin >> tmp;

variables.insert(std::pair<std::string, double> {RPN[i], tmp});

St.push\_back(tmp);

}

}

else St.push\_back(valid(RPN[i]));

}

}

return St.pop\_back();

}

double TPostfix::count(double\* variables, int number\_of\_variables) {

TStack<double> St;

std::map <std::string, double> usedvar;

int j = 0;

for (int i = 0; i < RPN.size(); i++) {

if (get\_prior(RPN[i])) {

if (RPN[i] == "+") St.push\_back(St.pop\_back() + St.pop\_back());

else if (RPN[i] == "-") St.push\_back(-St.pop\_back() + St.pop\_back());

else if (RPN[i] == "\*") St.push\_back(St.pop\_back() \* St.pop\_back());

else if (RPN[i] == "/") St.push\_back(1 / St.pop\_back() \* St.pop\_back());

else if (RPN[i] == "~") St.top() \*= -1;

else if (RPN[i] == "sin") St.push\_back(sin(St.pop\_back()));

else if (RPN[i] == "cos") St.push\_back(cos(St.pop\_back()));

else if (RPN[i] == "tan") St.push\_back(tan(St.pop\_back()));

else if (RPN[i] == "cot") St.push\_back(1 / tan(St.pop\_back()));

else if (RPN[i] == "exp") St.push\_back(exp(St.pop\_back()));

else St.push\_back(log(St.pop\_back()));

}

else {

if (RPN[i][0] == 'x') {

if (usedvar.count(RPN[i]))

St.push\_back(usedvar.find(RPN[i])->second);

else {

if (j <= number\_of\_variables - 1) {

usedvar.insert(std::pair<std::string, double> {RPN[i], variables[j++]});

St.push\_back(variables[j - 1]);

}

else throw std::invalid\_argument("Number of variables was entered incorrectly");

}

}

else St.push\_back(valid(RPN[i]));

}

}

return St.pop\_back();

}