МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

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

**(ННГУ)**

**Институт информационных технологий, математики и механики**

ЛАБОРАТОРНАЯ РАБОТА

на тему:

**«Постфиксная форма записи арифметических выражений»**

**Выполнил(а):** студент(ка) группы 3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Чистов А.Д./

Подпись

**Проверил:** к.т.н., доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д. /

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc153916400)

1 [Постановка задачи **Ошибка! Закладка не определена.**](#_Toc153916401)

[2 Руководство пользователя 5](#_Toc153916402)

[2.1 Приложение для демонстрации работы стека 5](#_Toc153916403)

[2.2 Приложение для демонстрации работы арифметического выражения 6](#_Toc153916404)

[3 Руководство программиста 7](#_Toc153916405)

[3.1 Описание алгоритмов 7](#_Toc153916406)

[3.1.1 Стек **Ошибка! Закладка не определена.**](#_Toc153916407)

[2.1.2 Арифметическое выражение **Ошибка! Закладка не определена.**](#_Toc153916408)

[3.2 Описание программной реализации 11](#_Toc153916409)

[2.2.1 Описание класса TStack 11](#_Toc153916410)

[2.2.2 Описание класса TArithmeticExpression 13](#_Toc153916411)

[Заключение 16](#_Toc153916412)

[Литература 17](#_Toc153916413)

[Приложения 18](#_Toc153916414)

[Приложение А. Реализация класса TStack 18](#_Toc153916415)

[Приложение Б. Реализация класса TArithmeticExpression 18](#_Toc153916416)

# Введение

Постфиксная форма, также известная как обратная польская запись, представляет собой удобный способ представления выражений, который может быть легко вычислен с использованием стека. Например, перевод в постфиксную форму может быть полезен при разработке компиляторов или интерпретаторов, где обратная польская запись может быть более удобной для обработки. Префиксная запись выражения требует, чтобы все операторы предшествовали двум операндам, с которыми они работают. Постфиксная, в свою очередь, требует, чтобы операторы шли после соответствующих операндов [1]. Перевод в постфиксную форму может упростить вычисление выражений и уменьшить необходимость использования скобок для определения порядка операций.

# Постановка задачи

Цель – реализовать классы TStack и TArithmeticExpression для преобразования инфиксного арифметического выражения в постфиксную форму.

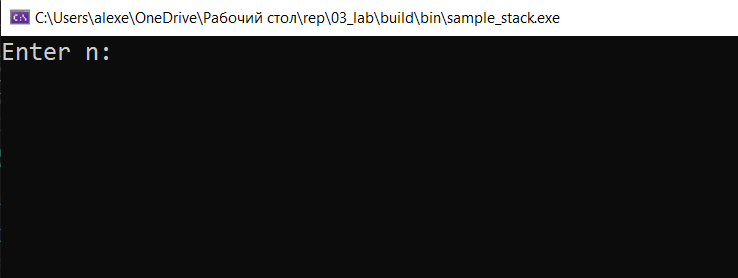
Задачи:

1. Исследовать тематическую литературу.
2. Реализовать класс TStack.
3. Реализовать класс TArithmeticExpression.
4. Провести тестирование разработанных классов для проверки их корректной работы.
5. Сделать выводы о проделанной работе.

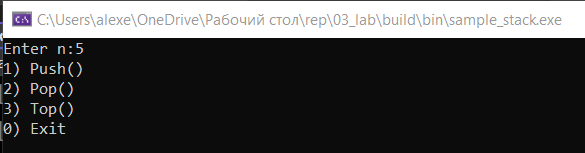
# Руководство пользователя

## Приложение для демонстрации работы векторов

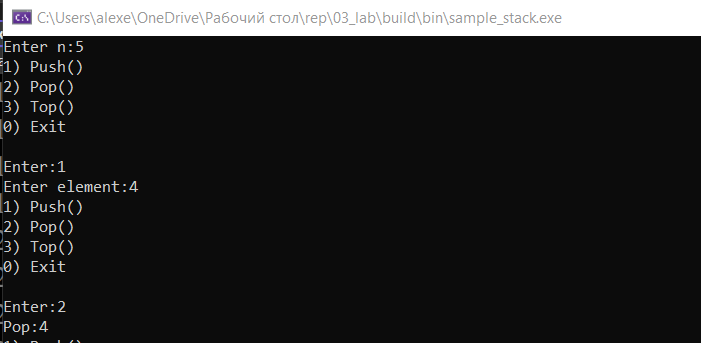
1. Запустите приложение с названием sample\_stack.exe. В результате появится окно, показанное ниже, где вам нужно будет ввести размерность стека (Рис. 1).



1. Основное окно программы
2. Далее вам нужно будет выбрать одно из следующих действий: 1) положить элемент в стек;2) вывести элемент на вершине стека;3) удалить верхний элемент из стека (Рис. 2).



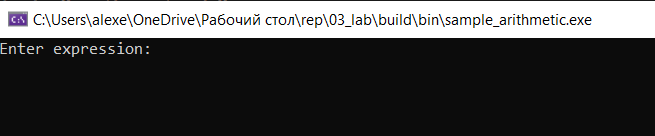
1. Функциональный набор приложения
2. Далее вы сможете наблюдать результат работы программы (Рис. 3).



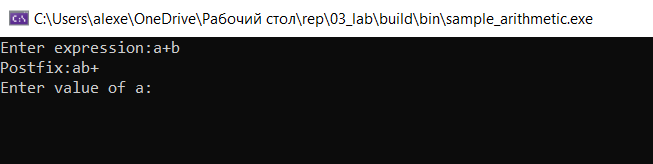
1. Результат работы программы

## Приложение для демонстрации работы матриц

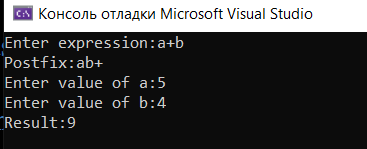
1. Запустите приложение с названием sample.arithmetic.exe, где вам нужно будет ввести ваше выражение в инфиксной форме. В результате появится окно, показанное ниже (Рис. 4).



1. Основное окно программы
2. Далее вам будет представлено ваше выражение в постфиксной форме и будет предложено ввести значение элементов (Рис. 5).



1. Постфиксная форма
2. Далее вы сможете наблюдать результат работы программы (Рис. 6).



1. Результат работы программы

# Руководство программиста

## Описание алгоритмов

### Стек

Стек – это структура хранения, основанная на принципе «Last in, first out», то есть мы можем взаимодействовать (добавлять, удалять) только с вершиной стека. Данный класс поддерживает следующие операции:

* **Операция добавления в стек**

Добавление элемента выполняется путем увеличения значения флага top, который указывает на вершину стека, и затем помещения нового элемента на позицию, указанную этим обновленным значением top. Если структура хранения еще не заполнена, то мы можем добавить элемент на позицию top+1.Пример:

До добавления элемента в стек:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 4 | 5 |  |  |

После добавления элемента в стек:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 4 | 5 | 6 |  |

* **Операция удаления с вершины**

Удаление элемента выполняется путем уменьшения значения флага top, который указывает на вершину стека, и затем удаления элемента на вершине стека. Если структура хранения еще не пуста, то мы можем удалить элемент на позицию top.

Пример:

До добавления элемента в стек:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 4 | 5 |  |  |

После добавления элемента в стек:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 4 |  |  |  |

* **Операция взятия элемента с вершины**

Взятие элемента с вершины выполняется при помощи флага top, который указывает на вершину стека. Если структура хранения еще не пуста, то мы можем взять элемент на позиции top.

Пример:

До добавления элемента в стек:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 4 | 5 |  |  |

Результат выполнения операции:5

* **Операция проверки на полноту(пустоту)**

Операция проверки на полноту(пустоту) проверяет, полон(пуст) ли стек в зависимости от значения флага.

Пример:

* полный стек:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 4 | 5 | 7 | 8 |

* пустой стек:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

### Арифметическое выражение

Арифметическое выражение –выражение, составленное из операндов,   
соединенных арифметическими операциями (+, -, \*, /). Программа основывается на использовании стека. Алгоритм ожидает на входе строку, представляющую математическое выражение, а также хэш-таблицу, где элементы соответствуют операндам в данном выражении.

Данный класс поддерживает следующие операции:

* **Получение постфиксной записи.**

Входные данные: строка, содержащая арифметическое выражение в инфиксной форме

Выходные данные: строка, содержащая арифметическое выражение в постфиксной форме

Пользователь вводит выражение в инфиксной форме, после чего алгоритм убирает лишние пробелы, проверяет на корректность, разделяет строку на лексемы и константы.

**Алгоритм:**

1. Создаем пустой стек операторов.

2. Создаем пустой массив для хранения постфиксной записи.

3. Проходим по каждому символу в инфиксной записи слева направо:

* Если символ является операндом, добавляем его в массив постфиксной записи.
* Если символ является открывающей скобкой, помещаем его в стек операторов.
* - Если символ является закрывающей скобкой, извлекаем операторы из стека и добавляем их в массив постфиксной записи до тех пор, пока не встретится открывающая скобка. Удаляем открывающую скобку из стека.
* - Если символ является оператором, извлекаем операторы из стека и добавляем их в массив постфиксной записи до тех пор, пока не будет найден оператор с меньшим или равным приоритетом. Затем помещаем текущий оператор в стек.

4. Извлекаем оставшиеся операторы из стека и добавляем их в массив постфиксной записи.

5.В результате получим выражение в постфиксной форме

Пример:

*Инфиксное выражение:* (a+b\*c)\*(c/d-e)

*Постфиксная запись*: abc\*+cd/e-\*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  | \* |
|  |  |  |  |  |  |  |  |  |  |  |  | - |
|  |  |  |  |  |  |  |  |  |  |  | e | e |
|  |  |  |  |  |  |  |  |  |  | / | / | / |
|  |  |  |  |  |  |  |  |  | d | d | d | d |
|  |  |  |  |  |  |  | c | c | c | c | c | c |
|  |  |  |  |  | + | + | + | + | + | + | + | + |
|  |  |  |  |  | \* | \* | \* | \* | \* | \* | \* | \* |
|  |  |  |  | c | c | c | c | c | c | c | c | c |
|  |  | b | b | b | b | b | b | b | b | b | b | b |
| a | a | a | a | a | a | a | a | a | a | a | a | a |

* Стек постфиксной формы:
* Cтек операторов:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | \* | \* |  |  |  | / | / | - | - |  |
|  |  | + | + | + | + |  | ( | ( | ( | ( | ( | ( |  |
| ( | ( | ( | ( | ( | ( | \* | \* | \* | \* | \* | \* | \* |  |

* **Вычисление значения выражения**

Входные данные: строка, содержащая арифметическое выражение в постфиксной форме, множество пар.

Выходные данные: вещественное значение, равное результату вычисления выражения.

**Алгоритм:**

1. Пока не достигнут конец входной последовательности:

* Если текущий символ - операнд, поместить его в стек.
* Если текущий символ - оператор, извлечь два последних операнда из стека, выполнить операцию над этими операндами и поместить результат обратно в стек.

1. Когда достигнут конец входной последовательности, результат вычисления будет находиться в стеке.

Пример:

1. *Инфиксная форма:(*a+b\*c)\*(c/d-e)
2. *Постфиксная форма:* abc\*+cd/e-\*

|  |  |
| --- | --- |
| Переменная | Значение |
| a | 1 |
| b | 2 |
| c | 3 |
| d | 4 |
| e | 5 |

Стек вычисления:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  | 3 |  |  | 4 | 5 |  |  |
|  | 2 | 2 | 6 | 3 | 3 | ¾ | -4,25 |  |
| 1 | 1 | 1 | 1 | 7 | 7 | 7 | 7 | -29,75 |

## Описание программной реализации

### Описание класса TStack

template <typename T>

class TStack {

private:

T\* elems;

int maxSize;

int top;

void Realloc(int step = \_step);

public:

TStack(int Size = \_maxSize);

TStack(const TStack<T>& stack);

virtual ~TStack();

T Pop();

T Top() const;

void Push(const T& elm);

bool IsEmpty(void) const;

bool IsFull(void) const;

};

Поля:

maxSize – количество доступной памяти, размер стека.

top – индекс верхнего элемента в стеке.

elems– память для представления стека.

Методы:

* TStack(int Size = \_maxSize);

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры: **maxSize** – количество выделяемой памяти.

* TStack(const TStack<T>& stack);

Назначение: конструктор копирования.

Входные параметры: **s** – экземпляр класса **TStack**, который нужно скопировать.

* virtual ~TStack();

Назначение: освобождение выделенной памяти.

* void Realloc(int step = \_step);

Назначение: перераспределение памяти.

Входные параметры: **step** – шаг выделяемой памяти.

* T Top() const;

Назначение: метод, возвращающий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: элемент, который находится на вершине стека.

* T Pop();

Назначение: метод, удаляющий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* void Push(const ValueType& e);

Назначение: метод, помещающий элемент на вершинку стека.

Входные параметры: **e** – элемент, который требуется добавить на вершину стека.

Выходные параметры: отсутствуют.

* bool isEmpty(void) const;

Назначение: проверка на пустоту стека.

Выходные параметры: **true** – стек пуст, **false** в противном случае.

* bool isFull (void) const;

Назначение: проверка на полноту стека.

Выходные параметры: **true** – стек стек заполнен, **false** в противном случае

### Описание класса TArithmeticExpression

class TArithmeticExpression {

private:

string infix;

vector<string> postfix;

vector<string> lexems;

static map<string, int> priority;

map<string, double> operands;

void Parse();

bool IsConst(const string& st) const;

bool IsOperator(char c) const;

bool IsParenthesis(char c) const;

bool IsDigitOrLetter(char c) const;

double Calculate(const map<string, double>& values);

void RemoveSpaces(string& str) const;

bool isCorrectInfixExpression();

public:

TArithmeticExpression(const string& \_infix);

string ToPostfix();

string GetInfix() { return infix; }

void SetValues();

void SetValues(const vector<double>& values);

double Calculate();

};

Поля:

Infix– выражение в инфиксной записи.

Postfix– выражение в посфиксной записи.

Lexems– набор лексем инфиксной записи

Priority– приоритет арифметических операндов

Operands– операнды и их значения

Методы:

* TArithmeticExpression (const string& \_infix);

Назначение: конструктор с параметрами.

Входные параметры:

Infix- инфиксное выражение

Выходные параметры: отсутствуют.

* string GetInfix();

Назначение: получение инфиксной формы.

Входные параметры отсутствуют.

Выходные параметры: инфиксная форма записи.

* string ToPostfix();

Назначение: получение постфиксной формы.

Входные параметры отсутствуют.

Выходные параметры: постфиксная форма записи.

* void SetValues();

Назначение: ввод с клавиатуры значений операндов

Входные параметры отсутствуют.

Выходные параметры: values- значения операндов.

* void SetValues(const vector<double>& values);

Назначение: установка значений операндов из вектора

Входные параметры: вектор значений.

Выходные параметры: values- значения операндов.

* double Calculate();

Назначение: метод вычисления выражения в постфиксной форме.

Выходные параметры: элемент, находящийся на верхушке стека, который является результатом вычислений.

* double Calculate (const map<string, double>& values);

Назначение: метод распознавания и выполнения арифметических операций.

Входные параметры: operator**\_** – символ оператора, a, b – операнды, над которыми выполнится операция.

Выходные параметры: результат выполнения соответствующей арифметической операции.

* void Parse ();

Назначение: подготовка к конвертированию инфиксной формы в постфиксную. Разделение инфиксной формы на лексемы.

Входные параметры отсутствуют.

Выходные параметры: отсутствуют

* bool IsConst(const char c) const;

Назначение: проверяет строку константа ли это.

Входные параметры:

c – оператор или операнд.

Выходные параметры: 1, если строка – это константа, 0 иначе.

* bool IsOperator(const char c) const;

Назначение: проверяет строку арифметический оператор ли это.

Входные параметры:

c – оператор или операнд.

Выходные параметры: 1, если строка – это арифметический оператор, 0 иначе.

* bool IsParenthesis(char c) const;

Назначение: проверяет строку скобка ли это.

Входные параметры:

c – открывающаяся или закрывающаяся скобка

Выходные параметры: 1, если строка – это скобка, 0 иначе.

* bool IsDigitOrLetter(char c) const;

Назначение: проверяет строку буква или цифра ли это.

Входные параметры:

c – буква или цифра

Выходные параметры: 1, если строка – это буква или цифра, 0 иначе.

* void RemoveSpaces(string& str) const;

Назначение: удаляет пробелы из строки

Входные параметры:

str – исходная строка

Выходные параметры: строка без пробелов

* bool isCorrectInfixExpression();

Назначение: проверяет инфикс на равенство открывающихся и закрывающихся скобок.

Входные параметры отсутствуют

Выходные параметры: 1, количество открывающихся и закрывающихся скобок равно, 0 иначе.

# Заключение

В результате данной лабораторной работы был разработан шаблонный класс стек, который поддерживает следующие операции: добавление элемента, удаление элемента с вершины стека, просмотр элемента на вершине стека без его удаления и другие. На его основе был мы разработан класс для работы с постфиксной формой записи арифметического выражения, который также поддерживает различные операции. В целом можно сказать, что проведенный анализ результатов показал, что использование постфиксной формы может быть очень полезным в решении определенных задач.

# Литература

1. Инфиксные, префиксные и постфиксные выражения [https://aliev.me/runestone/BasicDS/InfixPrefixandPostfixExpressions.html].
2. Лекция «Постфиксная форма» Сысоев А. В. [ <https://cloud.unn.ru/s/6g44ey6HFB4ncDy>].

# Приложения

## Приложение А. Реализация класса Tstack

template <typename T>

TStack<T>::TStack(int Size) {

if (Size <= 0) throw "maxSize must be bigger than 0.";

maxSize = Size;

top = -1;

elems = new T[maxSize];

}

template <typename T>

TStack<T>::TStack(const TStack<T>& stack) {

maxSize = stack.maxSize;

top = stack.top;

elems = new T[maxSize];

for (int i = 0; i <= top; i++) {

elems[i] = stack.elems[i];

}

}

template <typename T>

TStack<T>::~TStack() {

if (elems != NULL) {

delete[] elems;

top = -1;

maxSize = 0;

}

}

template <class T>

void TStack<T>::Realloc(int step)

{

if (step <= 0) { throw "Step must be greater than 0"; }

T\* tmp = new T[step + maxSize];

for (int i = 0; i < maxSize; i++) {

tmp[i] = elems[i];

}

delete[] elems;

elems = tmp;

maxSize += step;

}

template <typename T>

bool TStack<T>::IsFull(void) const {

return (top == maxSize - 1);

}

template <typename T>

bool TStack<T>::IsEmpty(void) const {

return (top == -1);

}

template <typename T>

T TStack<T>::Top() const {

if (IsEmpty()) { throw "Stack is empty."; }

return elems[top];

}

template <typename T>

void TStack<T>::Push(const T& elm) {

if (IsFull()) { Realloc(\_step); }

elems[++top] = elm;

}

template <typename T>

T TStack<T>::Pop() {

if (IsEmpty()) {throw "Stack is empty!";}

return elems[top--];

}

## Приложение Б. Реализация класса TArithmeticExpression

TArithmeticExpression::TArithmeticExpression(const string& \_infix) {

if (\_infix.empty()) {

throw("expression is empty");

}

infix = \_infix;

RemoveSpaces(infix);

if (!(isCorrectInfixExpression())) {

throw("non-correct number of parentheses");

}

}

map<string, int> TArithmeticExpression::priority = {

{"\*", 3},

{"/", 3},

{"+", 2},

{"-", 2},

{"(", 1},

{")", 1}

};

void TArithmeticExpression::RemoveSpaces(string& str) const {

str.erase(remove(str.begin(), str.end(), ' '), str.end());

}

bool TArithmeticExpression::IsConst(const string& s) const {

for (char c:s) {

if (!isdigit(c) && c != '.') {

return false;

}

}

return true;

}

bool TArithmeticExpression::IsOperator(char c) const {

return (c == '+' || c == '-' || c == '\*' || c == '/');

}

bool TArithmeticExpression::IsParenthesis(char c) const {

return (c == '(' || c == ')');

}

bool TArithmeticExpression::IsDigitOrLetter(char c) const{

return (isdigit(c) || c == '.' || isalpha(c));

}

void TArithmeticExpression::SetValues(){

double value;

for (auto& op : operands){

if (!IsConst(op.first)){

cout << "Enter value of " << op.first << ":";

cin >> value;

operands[op.first] = value;

}

}

}

void TArithmeticExpression::SetValues(const vector<double>& values) {

int i = 0;

for (auto& op : operands)

{

if (!IsConst(op.first)) {

operands[op.first] = values[i++];

}

}

}

void TArithmeticExpression::Parse()

{

string currentElement;

for (char c:infix) {

if (IsOperator(c) || IsParenthesis(c) || c == ' ') {

if (!currentElement.empty()) {

lexems.push\_back(currentElement);

currentElement = "";

}

lexems.push\_back(string(1, c));

}

else if (IsDigitOrLetter(c)) {

currentElement += c;

}

}

if (!currentElement.empty()) {

lexems.push\_back(currentElement);

}

}

string TArithmeticExpression::ToPostfix() {

Parse();

TStack<string> st;

string postfixExpression;

for (string item : lexems) {

if (item == "(") {

st.Push(item);

}

else if (item == ")") {

while (st.Top() != "(") {

postfixExpression += st.Top();

postfix.push\_back(st.Top());

st.Pop();

}

st.Pop();

}

else if (IsOperator(item[0])) {

while (!st.IsEmpty() && priority[item] <= priority[st.Top()]) {

postfixExpression += st.Top();

postfix.push\_back(st.Top());

st.Pop();

}

st.Push(item);

}

else {

double value = IsConst(item) ? stod(item) : 0.0;

operands.insert({ item, value });

postfix.push\_back(item);

postfixExpression += item;

}

}

while (!st.IsEmpty()) {

postfixExpression += st.Top();

postfix.push\_back(st.Top());

st.Pop();

}

return postfixExpression;

}

double TArithmeticExpression::Calculate(const map<string, double>& values)

{

for (auto& val : values) {

try {

operands.at(val.first) = val.second;

}

catch (out\_of\_range& e) {}

}

TStack<double> st;

double leftOperand, rightOperand;

for (string lexem : postfix) {

if (lexem == "+") {

rightOperand = st.Top();st.Pop();

leftOperand = st.Top();st.Pop();

st.Push(leftOperand + rightOperand);

}

else if (lexem == "-") {

rightOperand = st.Top();st.Pop();

leftOperand = st.Top();st.Pop();

st.Push(leftOperand - rightOperand);

}

else if (lexem == "\*") {

rightOperand = st.Top();st.Pop();

leftOperand = st.Top();st.Pop();

st.Push(leftOperand \* rightOperand);

}

else if (lexem == "/") {

rightOperand = st.Top();st.Pop();

leftOperand = st.Top();st.Pop();

if (rightOperand == 0) {throw"Error";}

st.Push(leftOperand / rightOperand);

}

else {

st.Push(operands[lexem]);

}

}

return st.Top();

}

double TArithmeticExpression::Calculate() {

return Calculate(operands);

}

bool TArithmeticExpression::isCorrectInfixExpression()

{

int count = 0;

for (char c : infix)

{

if (c == '(') count++;

else if (c == ')') count--;

if (count < 0) return false;

}

return (count == 0);

}