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:

**«Вычисление арифметических выражений»**

**Выполнил:** студент группы 3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Шкребко М.С.

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc154754475)

[1 Постановка задачи 4](#_Toc154754476)

[2 Руководство пользователя 4](#_Toc154754477)

[2.1 Приложение для демонстрации работы стека 4](#_Toc154754478)

[2.2 Приложение для демонстрации работы перевода арифметического выражения в постфиксную запись 5](#_Toc154754479)

[3 Руководство программиста 7](#_Toc154754480)

[3.1 Описание алгоритмов 7](#_Toc154754481)

[3.1.1 Стек 7](#_Toc154754482)

[3.1.2 Перевод в постфиксную запись 8](#_Toc154754483)

[3.2 Описание программной реализации 8](#_Toc154754484)

[3.2.1 Описание класса Stack 8](#_Toc154754485)

[3.2.2 Описание класса TArithmeticExpression 10](#_Toc154754486)

[Заключение 13](#_Toc154754487)

[Литература 14](#_Toc154754488)

[Приложения 15](#_Toc154754489)

[Приложение А. Реализация класса Stack 15](#_Toc154754490)

[Приложение Б. Реализация класса TArithmeticExpression 16](#_Toc154754491)

# Введение

Целью данной лабораторной работы является создание стека для того чтобы можно было реализовать обратную польскую запись. Для этого надо реализовать шаблонный класс с нужными методами.

Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO.

Применение стека упрощает и ускоряет работу программы, так как идет обращение к нескольким данным по одному адресу.

Обратная польская запись — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций.

Отличительной особенностью обратной польской нотации является то, что все аргументы расположены перед знаком операции.

Постфиксная форма используется для оптимизации вычислений, так как помогает избежать проблем с приоритетом операторов и порядков операций, делая выражения более однозначными.

# Постановка задачи

Цель – реализовать шаблонный класс для представления стека, и на его основе реализовать класс, содержащий функции для представления арифметического выражения с его постфиксной формой.

Задачи при реализации второго класса:

1. Изучение основных принципов работы со стеком.
2. Изучение правил преобразования инфиксного выражения в постфиксное.
3. Написание программы на С++, использующей стек для преобразования арифметического выражения.
4. Тестирование программы на различных входных данных, включая выражения с разными операциями и скобками.

# Руководство пользователя

## Приложение для демонстрации работы стека

1. Запустите приложение с названием sample\_tstack.exe. В результате появится окно, показанное ниже и вам будет предложено ввести размер максимальный стека (рис. 1).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

1. Основное окно программы
2. Далее будет предоставлено окно для изменения стека в котором можно добавить элемент, удалить, взять верхний, проверить на полноту и пустоту (рис. 2).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Выбор действия

## Приложение для демонстрации работы перевода арифметического выражения в постфиксную запись

1. Запустите приложение с названием sample\_tarithmetic.exe. В результате появится окно, показанное ниже, вам будет предложено ввести арифметическое выражение. (рис. 3).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

1. Основное окно программы
2. Нужно после ввода выражения ввести значение операндов и будет выведена постфиксная форма и итог выражения (рис. 4).

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

1. Результат тестирования

# Руководство программиста

## Описание алгоритмов

### Стек

Стек представляет собой абстрактную структуру данных, организованную по принципу "последний вошел - первый вышел". Это означает, что элементы добавляются и удаляются только с одного конца стека, который называется вершиной.

Операции, которые можно выполнять со стеком, включают добавление элемента на вершину стека и удаление элемента с вершины стека. При этом доступ к остальным элементам стека осуществлять нельзя, кроме верхнего элемента.

**Операция получения размера стека**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 |  |

Длина стека равна 4.

**Операция добавления в стек**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 |  |

Добавим элемент 13

Результат:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 | 13 |

**Операция удаления с вершины**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 | 13 |

Результат:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 |  |

**Операция проверки на пустоту**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 | 13 |

Результат: false – стек не пуст

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

Результат: true – стек пуст

**Операция проверки на полноту**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 | 13 |

Результат: true – стек полон

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 |  |

Результат: false – стек не полон

**Операция проверки последнего элемента**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 12 | 3 | 2 | 13 |

Результат: 13

### Перевод в постфиксную запись

Входные данные: арифметическое выражение.

Выходные данные: постфиксная форма.

Алгоритм:

1. Для операций вводится приоритет:

3: \*, /

2: +, -

1: (, )

1. Для хранения данных используются два стека: первый для хранения операндов и результатов, второй для хранения операций.
2. Исходное выражение просматривается слева на право пока не дойдем до его конца.
   1. Операнды по мере их появления помещаются в стек 1.
   2. Символы операции ( помещаются в стек 2 по следующему правилу:
      1. При появлении ) последовательно изымаются элементы (операции) из стека 2 и переносятся в стек 1 пока стек 2 не пуст или не обнаружена (.
      2. Левая скобка кладется в стек 2.
      3. Если текущая операция, полученная при обходе выражения имеет более низкий приоритет операции, чем на вершине стека 2, то все операции, приоритет которых больше или равен приоритету текущей операции перекладываются из стека 2 в стек 1, текущая операция переносится в стек 2.
3. По завершению выражения операции из стека 2 перекладываются в стек 1, пока он не пуст.

### Алгоритм вычисления значения выражения, записанного в постфиксной форме

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

Выходные данные: вещественное значение, равное результату вычисления выражения.

Алгоритм:

1. Пока не достигнут конец входной последовательности, читаем очередную лексему.
   1. Если прочитан операнд, то его значение помещается в стек.
   2. Если прочитана операция, то из стека изымаются значения двух последних операндов, выполняется операция над этими операндами. Результат выполнения операции кладется обратно в стек.
2. Если достигнут конец входной последовательности, то в стеке хранится результат вычисления.

## Описание программной реализации

### Описание класса Stack

template <typename T> class Stack {

private:

int top;

int maxSize;

T\* elems;

public:

Stack(int size = 10);

Stack(const Stack<T>& s);

~Stack();

void Pop();

T Top() const;

void Push(const T e);

bool IsFull()const { return top == maxSize - 1; }

bool IsEmpty()const { return top == -1; }

};

Назначение: представление стека.

Поля:

maxSize – максимальный размер стека.

\*elems – память для представления элементов стека.

top – индекс вершины стека (-1, если стек пустой).

Методы:

Stack(int size = 10);

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры:

size – максимальный размер стека

Выходные параметры: отсутствуют.

Stack(const Stack<T>& s);

Назначение: конструктор копирования.

Входные параметры:

s – стек,на основе которого создаем новый стек.

Выходные параметры: отсутствуют.

~Stack();

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

T Top() const;

Назначение: получение элемента, находящийся в вершине стека.

Входные параметры отсутствуют.

Выходные параметры: элемент с вершины стека, последний добавленный элемент.

bool IsEmpty()const;

Назначение: проверка на пустоту.

Входные параметры: отсутствуют.

Выходные параметры: 1, если стек пуст, 0 иначе.

bool IsFull()const;

Назначение: проверка на полноту.

Входные параметры: отсутствуют.

Выходные параметры: 1, если стек полон, 0 иначе.

void Push(const T e);

Назначение: добавление элемента в стек.

Входные параметры:

e – элемент, который добавляем.

Выходные параметры отсутствуют.

void Pop();

Назначение: удаление элемента из вершины стека.

Входные параметры отсутствуют.

### Описание класса TArithmeticExpression

class TArithmeticExpression {

private:

string infix;

string spostfix;

vector<string> postfix;

vector<string> lexems;

static map<string, int> priority;

map<string, double> operands;

void Parse();

void ToPostfix();

void InToPostfix();

bool IsOperator(const string& lexem) const;

bool isOperatorChar(char c) const;

bool IsConst(const string& lexem) const;

public:

TArithmeticExpression(const string& infix);

string GetInfix() const { return infix; }

string GetPostFix\_str() const { return spostfix; }

vector<string> GetPostfix() const { return postfix; }

bool isCorrectInfixExpression();

void SetValues();

double Calculate();

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

void ShowPostfix();

};

Назначение: работа с инфиксной формой записи арифметических выражений

Поля:

infix – выражение в инфиксной записи.

postfix – выражение в посфиксной записи.

spostfix - строка постфиксного арифметического выражения.

lexems – набор лексем инфиксной записи

priority – приоритет арифметических операндов

operands – операнды и их значения

Методы:

TArithmeticExpression(const string& infix)

Назначение: инициализация полей класса **TArithmeticExpression**.

Входные параметры: infix – арифметическое выражение в инфиксной форме.

Выходные параметры: нет.

void Parse()

Назначение: разбиение строки инфиксной формы арифметического выражения.

Входные параметры: нет.

Выходные параметры: нет.

void ToPostfix();

Назначение: перевод арифметического выражения в постфиксную форму.

Входные параметры: нет.

Выходные параметры: нет.

**void InToPostfix();**

Назначение: запись постфиксной форм арифметического выражения в строку.

Входные параметры: нет.

Выходные параметры: нет.

bool IsOperator(const string& lexem) const

Назначение: проверка является ли строка оператором.

Входные параметры: константная ссылка на строку.

Выходные параметры: 1- если сторка- оператор, 0- иначе.

bool IsConst(const string& lexem) const

Назначение: проверка является ли строка числом.

Входные параметры: константная ссылка на строку.

Выходные параметры: 1- если сторка- число, 0- иначе.

string GetInfix() const

Назначение: получение инфиксной формы арифметического выражения.

Входные параметры: нет.

Выходные параметры: **infix** – инфиксная запись выражения.

vector<string> GetPostfix();

Назначение: получение постфиксной формы арифметического выражения.

Входные параметры: нет.

Выходные параметры: постфиксная запись выражения.

string GetPostfix\_str() const;

Назначение: получение строки постфиксной формы арифметического выражения.

Входные параметры: нет.

Выходные параметры: постфиксная запись выражения.

bool isCorrectInfixExpression()

Назначение: проверка инфиксного выражения на корректность.

Входные параметры: нет.

Выходные параметры: 1 – выражение корректно, иначе 0.

void SetValues()

Назначение: присваивание значений операндам.

Входные параметры: нет.

Выходные параметры: нет.

void ShowPostfix()

Назначение: вывод постфиксной формы арифметического выражения на экран.

Входные параметры: нет.

Выходные параметры: нет.

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

Назначение: вычисление результата арифметического выражения.

Входные параметры: values – список операндов и их значений.

Выходные параметры: результат вычисления.

double Calculate()

Назначение: вычисление результата арифметического выражения.

Входные параметры: нет.

Выходные параметры: результат вычисления.

# Заключение

В рамках работы был разработан шаблонный класс для реализации стека, который поддерживает основные операции, такие как добавление, удаление и получения элемента с верхушки стека, методы проверки на полноту и пустоту стека. Также был разработан класс для реализации алгоритмов перевода выражения в постфиксную форму и вычисления её значения.

# Литература

1. Польская запись [https://ru.wikipedia.org/wiki/Польская\_запись].

# Приложения

## Приложение А. Реализация класса Stack

#ifndef \_STACK\_H

#define \_STACK\_H

#include <iostream>

using namespace std;

template <typename T> class Stack {

private:

int top;

int maxSize;

T\* elems;

public:

Stack(int size = 10);

Stack(const Stack<T>& s);

~Stack();

void Pop();

T Top() const;

void Push(const T e);

bool IsFull()const { return top == maxSize - 1; }

bool IsEmpty()const { return top == -1; }

};

template <typename T>

Stack<T>::Stack(int size) {

if (size <= 0) {

throw "Negative or zero size!";

}

top = -1;

maxSize = size;

elems = new T[maxSize];

}

template <typename T>

Stack<T>::Stack(const Stack <T>& s) {

top = s.top;

maxSize = s.maxSize;

elems = new T[maxSize];

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

elems[i] = s.elems[i];

}

}

template <typename T>

Stack<T>::~Stack() {

delete[] elems;

maxSize = 0;

top = -1;

}

template <typename T>

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

if (IsEmpty()) {

throw "Stack is empty!";

}

else {

return elems[top];

}

}

template <typename T>

void Stack<T>::Push(const T e) {

if (IsFull()) {

throw "Stack is full!";

}

else {

elems[++top] = e;

}

}

template <typename T>

void Stack<T>::Pop() {

if (IsEmpty()) {

throw "Stack is empty!";

}

top--;

}

#endif

## Приложение Б. Реализация класса TArithmeticExpression

#include "tarithmetic.h"

void TArithmeticExpression::Parse()

{

string currentElement;

for (char c : infix) {

if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '(' || c == ')') {

if (!currentElement.empty()) {

lexems.push\_back(currentElement);

currentElement = "";

}

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

}

else if (isdigit(c) || c == '.') {

currentElement += c;

}

else if (isalpha(c)) {

currentElement += c;

}

}

if (!currentElement.empty()) {

lexems.push\_back(currentElement);

}

}

void TArithmeticExpression::ToPostfix() {

Parse();

Stack<string> st;

string stackItem;

for (string item : lexems) {

if (item == "(") {

st.Push(item);

}

else if (item == ")") {

stackItem = st.Top();

st.Pop();

while (stackItem != "(") {

postfix.push\_back(stackItem);

stackItem = st.Top();

st.Pop();

}

}

else if (item == "+" || item == "-" || item == "\*" || item == "/") {

while (!st.IsEmpty()) {

stackItem = st.Top();

st.Pop();

if (priority[item] <= priority[stackItem])

postfix.push\_back(stackItem);

else {

st.Push(stackItem);

break;

}

}

st.Push(item);

}

else {

double value = 0.0;

if (IsConst(item)) {

value = stod(item);

}

operands.insert({ item, value });

postfix.push\_back(item);

}

}

while (!st.IsEmpty()) {

stackItem = st.Top();

st.Pop();

postfix.push\_back(stackItem);

}

}

bool TArithmeticExpression::IsOperator(const string& lexem) const

{

bool flag = false;

for (const auto& c : priority)

{

if (lexem == c.first)

{

flag = true;

}

}

return flag;

}

bool TArithmeticExpression::IsConst(const string& lexem) const

{

bool flag = true;

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

{

if (!isdigit(lexem[i]))

{

flag = false;

}

}

return flag;

}

TArithmeticExpression::TArithmeticExpression(const string& infix) :infix(infix) {

ToPostfix();

InToPostfix();

}

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

{"(",1},{")",1},{"+",2},{"-",2}, {"\*",3},{"/",3}

};

bool TArithmeticExpression::isCorrectInfixExpression()

{

int openParentheses = 0;

int closeParentheses = 0;

for (char c : infix)

if (c == '(')

openParentheses++;

else if (c == ')')

closeParentheses++;

return openParentheses == closeParentheses;

}

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;

}

}

}

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

{

for (auto& val : values) {

try {

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

}

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

}

Stack<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);

}

void TArithmeticExpression::ShowPostfix()

{

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

{

cout << postfix[i] << " ";

}

cout << endl;

}

void TArithmeticExpression::InToPostfix()

{

for (const auto& elem : postfix) {

spostfix += elem;

}

}