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:** студент группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Табунов А. И./

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc153624766)

[1 Постановка задачи 4](#_Toc153624767)

[2 Руководство пользователя 5](#_Toc153624768)

[2.1 Приложение для демонстрации работы стека 5](#_Toc153624769)

[2.2 Приложение для демонстрации работы перевода арифметического выражения в постфиксную запись 6](#_Toc153624770)

[3 Руководство программиста 8](#_Toc153624771)

[3.1 Описание алгоритмов 8](#_Toc153624772)

[3.1.1 Стек 8](#_Toc153624773)

[3.1.2 Арифметическое выражение 9](#_Toc153624774)

[3.2 Описание программной реализации 12](#_Toc153624775)

[3.2.1 Описание класса TStack 12](#_Toc153624776)

[3.2.2 Описание класса Expression 13](#_Toc153624777)

[Заключение 17](#_Toc153624778)

[Литература 18](#_Toc153624779)

[Приложения 19](#_Toc153624780)

[Приложение А. Реализация класса TStack 19](#_Toc153624781)

[Приложение Б. Реализация класса Expression 20](#_Toc153624782)

# Введение

Лабораторная работа направлена на изучение алгоритма преобразования математических выражений из инфиксной записи в постфиксную (обратную польскую) запись. Инфиксная запись — это традиционный способ записи математических выражений, где операторы расположены между операндами. Постфиксная запись, наоборот, предполагает расположение операторов после соответствующих операндов.

В данной лабораторной работе студенты будут изучать основные принципы работы алгоритма преобразования инфиксной записи в постфиксную и реализовывать его на практике. Это позволит им лучше понять принципы работы стека и освоить навыки работы с алгоритмами обработки строк и вычисления математических выражений.

# Постановка задачи

**Цель:**

Реализовать шаблонный класс TStack. Используя класс TStack реализовать класс перевода арифметического выражения в постфиксную форму Expression. Научиться использовать стек для преобразования инфиксного (обычного) арифметического выражения в постфиксную (обратную польскую) форму.

**Задачи:**

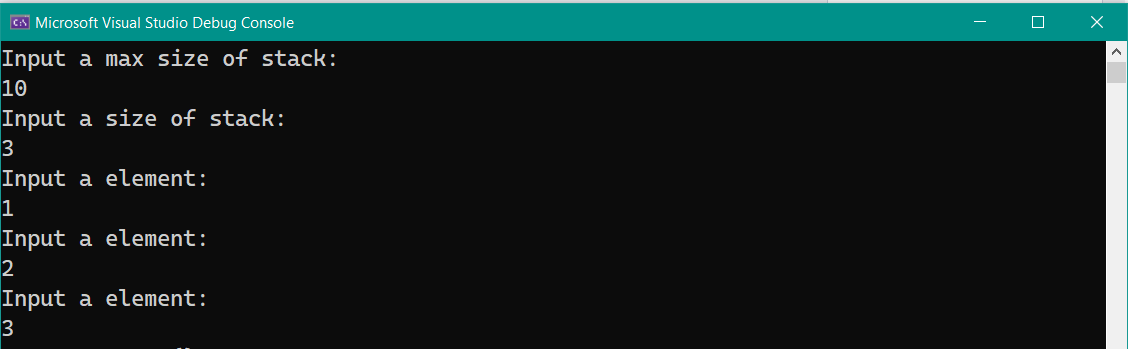
1. Изучение основных принципов работы со стеком.
2. Изучение правил преобразования инфиксного выражения в постфиксное.
3. Написание программы на С++, использующей стек для преобразования арифметического выражения.
4. Анализ времени выполнения программы и оценка эффективности использования стека для данной задачи.
5. Тестирование программы на различных входных данных, включая выражения с разными операциями и скобками.

В результате лабораторной работы студент должен освоить принципы работы со стеком, понять преимущества использования постфиксной формы для вычисления арифметических выражений и научиться применять их на практике.

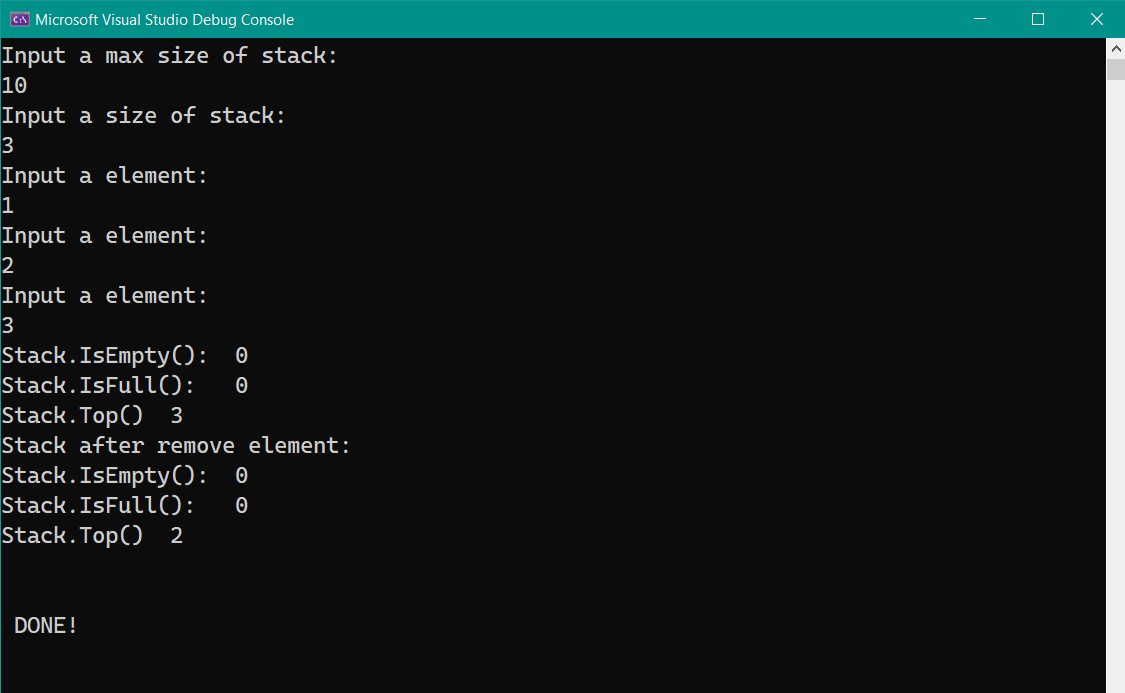
# Руководство пользователя

## 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, число элементов в стеке и сами целочисленные элементы. (рис. 1).



1. Основное окно программы
2. После ввода будет выведены результаты соответствующих операций и функций стека. Если введенный размер равен нулю, то метод Top() не вызывается.(рис. 2).



1. Результат тестирования функций класса TStack

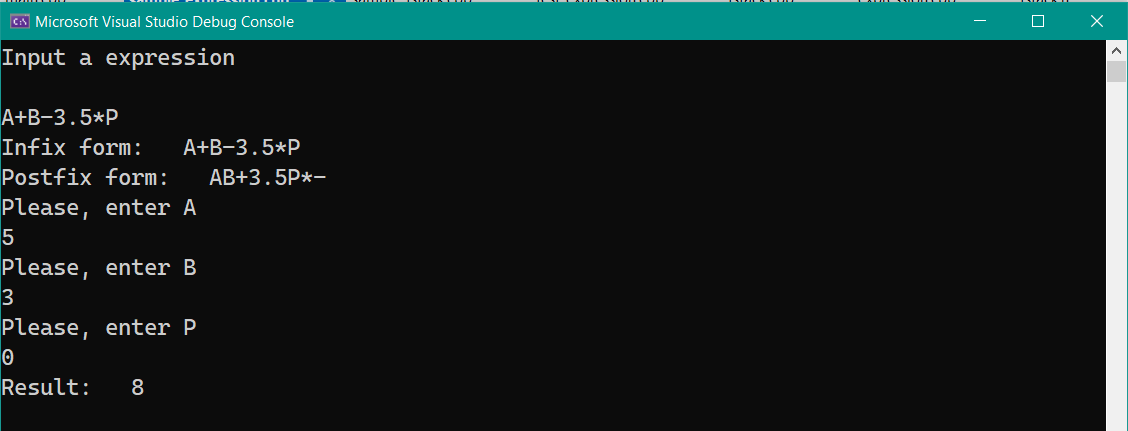
## Приложение для демонстрации работы перевода арифметического выражения в постфиксную запись

1. Запустите приложение с названием sample\_expression.exe. В результате появится окно, показанное ниже, вам будет предложено ввести арифметическое выражение. После ввода арифметического выражения необходимо ввести количество используемых в нем переменных, далее ввести каждый из них

(рис. 3).



1. Основное окно программы
2. После ввода арифметического выражения будет выведены результаты соответствующих операций и функций (рис. 4).



1. Результат тестирования функций класса Expression

# Руководство программиста

## Описание алгоритмов

### Стек

Стек – это структура хранения, основанная на принципе «Last in, first out». Операции, доступные с данной структурой хранения, следующие: добавление элемента в вершину стека, удаление элемента из вершины стека, взять элемент с вершины стека, проверка на полноту, проверка на пустоту.

**Операция добавления элемента в вершину стека**

Операция добавления элемента реализуется при помощи флага, указывающий на последний занятый элемент ( на вершину стека ). Если структура хранения ещё не полна, то мы можем добавить элемент на top+1 место.

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 2 |  |  |

Операция добавления элемента (1) в вершину:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 4 | 2 |  |

**Операция удаления элемента из вершины стека**

Операция удаления элемента реализуется при помощи флага, указывающий на последний занятый элемент ( на вершину стека ) . Если структура хранения ещё не пуста, то мы можем удалить элемент с индексом top .

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 2 |  |  |

Операция добавления элемента (1) в вершину:

|  |  |  |  |
| --- | --- | --- | --- |
| 2 |  |  |  |

**Операция взятия элемента с вершины.**

Операция взятия элемента с вершины также реализуется при помощи флага, указывающий на последний занятый элемент ( на вершину стека ). Если структура хранения не пуста, мы можем взять элемент с вершины.

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 2 |  |  |

Операция взятия элемента с вершины стека:

Результат: 4

**Операция проверки на полноту.**

Операция проверки на полноту проверяет, полон ли стек. Также реализуется при помощи флага, указывающий на вершину стека.

Пример 1:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 2 |  |  |

Операция проверки на полноту:

Результат: false

Пример 2:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 2 | 2 | 2 |

Операция проверки на полноту:

Результат: true

**Операция проверки на пустоту.**

Операция проверки на полноту проверяет, есть ли хотя бы один элемент в стеке. Также реализуется при помощи флага, указывающий на вершину стека.

Пример 1:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 2 |  |  |

Операция проверки на полноту:

Результат: false

Пример 2:

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

Операция проверки на полноту:

Результат: true

### Арифметическое выражение

Программа предоставляет возможности для работы с арифметическими выражениями: получение инфиксной записи, получение постфиксной записи, получение результата.

Алгоритм на входе требует строку, которая представляет некоторое арифметическое выражение, и хэш-таблицу, элементы которой представляют операнды в арифметическом выражении. Алгоритм также вводит приоритет арифметических операций согласно математическим правилам: скобки, умножение/деление, сложение/вычитание.

**Получение инфиксной записи.**

Функция просто выведет исходную строку в инфиксной записи.

**Получение постфиксной записи.**

Изначально алгоритм подготавливает выражение: убирает лишние пробелы, проверяет на корректность введенных данных, разделяет строку на операции и операнды. Таким образом, до начала перевода в постфиксную форму в программе уже есть разделенный набор операций и операндов.

Алгоритм:

1. Создаем пустой стек операторов.

2. Создаем пустой массив для хранения постфиксной записи.

3. Проходим по каждому символу в инфиксной записи слева на право:

- 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.

После завершения алгоритма массив постфиксной записи будет содержать инфиксное выражение в постфиксной форме.

Пример:

Выражение: А + (В – C) \* D

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  | + |
|  |  |  |  |  |  |  |  | \* |
|  |  |  |  |  |  |  |  | D |
|  |  |  |  |  |  | - | - | - |
|  |  |  |  |  | C | C | C | C |
|  |  |  | B | B | B | B | B | B |
| A | A | A | A | A | A | A | A | A |

Стек:

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

**Вычисление результата.**

Алгоритм вычисления значения выражения в постфиксной записи (обратной польской записи) выглядит следующим образом:

1. Создаем пустой стек для хранения операндов.

2. Проходим по каждому символу в постфиксной записи:

- Если символ является операндом, помещаем его в стек операндов.

- Если символ является оператором, извлекаем два операнда из стека, применяем оператор к этим операндам и помещаем результат обратно в стек.

3. После завершения прохода по всем символам, результат вычисления будет находиться на вершине стека операндов.

Полученное значение на вершине стека будет являться результатом вычисления постфиксной записи.

Пример:

Выражение: А + (В – C) \* D

Постфиксная запись: ABC-D\*+

Значение операндов: A = B = 3, C = D = 2

Стек:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  | 3 | 3 | 1 | 1 | 2 |  |
| 3 | 3 | 3 | 3 | 3 | 3 | 2 | 4 |

## Описание программной реализации

### Описание класса TStack

template <class Type>

class TStack {

private:

int maxSize;

int top;

Type\* elems;

public:

TStack(int maxSize1 = 100);

TStack(const TStack<Type>& s);

~TStack();

Type Top() const;

bool IsEmpty() const;

bool IsFull() const;

void Push(const Type& elem);

void Pop();

};

Назначение: представление стека .

Поля:

maxSize – максимальный размер стека.

\*elems – память для представления элементов стека.

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

Методы:

TStack(int maxSize1 = 100);

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры:

maxSize1 – максимальный размер стека (по умолчанию 100).

Выходные параметры: отсутствуют.

TStack(const TStack<Type>& s);

Назначение: конструктор копирования.

Входные параметры:

s – стек, на основе которого создаем новый стек.

Выходные параметры: отсутствуют.

~TStack();

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

**Type 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 Type& elem);

Назначение: добавление элемента в стек.

Входные параметры:

elem – элемент, который добавляем.

Выходные параметры отсутствуют.

void pop();

Назначение: удаление элемента из вершины стека.

Входные параметры отсутствуют.

### Описание класса Expression

class Expression

{

private:

string infix;

vector<string> postfix;

vector<string> lexems;

map<string, int> priority;

map<string, double> operands;

void Parse();

void ToPostfix();

bool IsOperator(const string& lecsem) const;

bool IsConst(const string& lecsem) const;

public:

TArithmeticExpression(string infix);

string GetInfix() const { return infix; }

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

void ShowPostfix();

bool isCorrectInfixExpression();

vector<string> GetOperands() const;

void SetValues();

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

double Calculate();

};

Назначение: представление арифметического выражения.

Поля:

**infix**- инфиксная форма арифметического выражения.

**postfix**- постфиксная форма арифметического выражения.

**lexems**- инфиксная форма арифметического выражения с выделенными лексемами

**priority**- возможные операции с приоритетами.

**operands**- список всех операндов арифметического выражения.

Методы:

Expression(string infix)

Назначение: инициализация полей класса **Expression**.

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.

bool IsOperator(const string& lecsem) 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& lecsem) 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.

void ShowPostfix()

Назначение: вывод постфиксной формы арифметического выражения на экран.

Входные параметры: нет.

Выходные параметры: нет.

bool isCorrectInfixExpression()

Назначение: проверка инфиксного выражения на корректность.

Входные параметры: нет.

Выходные параметры: 1 – выражение корректно, иначе 0.

vector<string> GetOperands() const

Назначение: получение всех операндов арифметического выражения.

Входные параметры: нет.

Выходные параметры: вектор операндов арифметического выражения.

void SetValues()

Назначение: присваивание значений операндам.

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 ходе выполнения лабораторной работы было изучены основные принципы работы алгоритма преобразования математических выражений из инфиксной записи в постфиксную. Также были получено практические навыки реализации этого алгоритма и работы с постфиксной записью.

Изучение данного алгоритма позволило студентам лучше понять принципы работы стека, освоить навыки работы с алгоритмами обработки строк и вычисления математических выражений. Также они узнали о преимуществах постфиксной записи перед инфиксной и научились применять её в практических задачах.

Таким образом, выполнение лабораторной работы позволило студентам расширить свои знания в области алгоритмов обработки математических выражений и приобрести навыки работы с постфиксной записью. Эти знания и навыки будут полезны им в дальнейшем образовании и профессиональной деятельности.

# Литература

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

# Приложения

## Приложение А. Реализация класса TStack

#ifndef \_TSTACK\_H

#define \_TSTACK\_H

using namespace std;

template <class Type>

class TStack {

private:

int maxSize;

int top;

Type\* arr;

public:

TStack(int maxSize = 100);

TStack(const TStack<Type>& s);

~TStack();

Type Top() const;

bool IsEmpty() const;

bool IsFull() const;

void Push(const Type& elem);

void Pop();

};

template <class Type>

TStack<Type>::TStack(int maxSize)

{

if (maxSize < 1)

throw exception("Size should be > 0");

this->maxSize = maxSize;

arr = new Type[maxSize];

top = -1;

}

template <class Type>

TStack<Type>::TStack(const TStack<Type>& s)

{

maxSize = s.maxSize;

top = s.top;

arr = new Type[maxSize];

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

{

arr[i] = s.arr[i];

}

}

template <class Type>

TStack<Type>::~TStack()

{

delete[] arr;

}

template <class Type>

bool TStack<Type>::IsFull() const

{

return top + 1 >= maxSize;

}

template <class Type>

bool TStack<Type>::IsEmpty() const

{

return top <= -1;

}

template <class Type>

Type TStack<Type>::Top() const

{

if (top <= -1)

throw exception("Stack is empty");

return arr[top];

}

template <class Type>

void TStack<Type>::Push(const Type& elem)

{

if (top + 1 >= maxSize)

throw exception("Stack is full");

arr[++top] = elem;

}

template <class Type>

void TStack<Type>::Pop()

{

if (top == -1) {

throw exception("Stack is empty");

}

top--;

}

#endif // !\_TSTACK\_H

## Приложение Б. Реализация класса Expression

**#include "expression.h"**

**#include <iostream>**

**#include <string>**

**#include <map>**

**Expression::Expression(const string& infx) : infix(infx)**

**{**

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

**ToPostfix();**

**InToPostfix();**

**}**

**void Expression::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);**

**}**

**}**

**vector<string> Expression::GetOperands() const**

**{**

**vector<string> op;**

**for (const auto& item : operands)**

**if (!IsConst(item.first))**

**{**

**op.push\_back(item.first);**

**}**

**return op;**

**}**

**bool Expression::IsOperator(const string& lecsem) const**

**{**

**bool flag = false;**

**for (const auto& c : priority)**

**{**

**if (lecsem == c.first)**

**{**

**flag = true;**

**break;**

**}**

**}**

**return flag;**

**}**

**void Expression::SetValues()**

**{**

**double value;**

**for (auto& op : operands)**

**{**

**if (!IsConst(op.first))**

**{**

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

**cin >> value;**

**operands[op.first] = value;**

**}**

**}**

**}**

**bool Expression::IsConst(const string& lecsem) const**

**{**

**bool hasDot = false;**

**bool flag = true;**

**for (char c : lecsem) {**

**if (!std::isdigit(c)) {**

**if (c == '.' && !hasDot) {**

**hasDot = true;**

**}**

**else {**

**flag = false;**

**break;**

**}**

**}**

**}**

**return flag;**

**}**

**void Expression::ToPostfix() {**

**Parse();**

**TStack<string> st(lexems.size());**

**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);**

**}**

**}**

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

**{**

**for (auto & val : values)**

**{**

**operands[val.first] = val.second;**

**}**

**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 {**

**try**

**{**

**st.Push(operands.at(lexem));**

**}**

**catch (const std::out\_of\_range& e)**

**{**

**std::cout << "Please, enter " << lexem << std::endl;**

**double value;**

**std::cin >> value;**

**operands[lexem] = value;**

**st.Push(value);**

**}**

**}**

**}**

**return st.Top();**

**}**

**double Expression::Calculate()**

**{**

**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 {**

**try**

**{**

**st.Push(operands.at(lexem));**

**}**

**catch (const std::out\_of\_range& e)**

**{**

**std::cout << "Please, enter " << lexem << std::endl;**

**double value;**

**std::cin >> value;**

**operands[lexem] = value;**

**st.Push(value);**

**}**

**}**

**}**

**return st.Top();**

**}**

**bool IsOperandNotScobes(char c) {**

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

**}**

**bool Expression::isCorrectInfixExpression()**

**{**

**int openParentheses = 0;**

**int closeParentheses = 0;**

**if (infix[0] == '\*' || infix[0] == '/' || infix[0] == '+' || infix[0] == ')')**

**{**

**return false;**

**}**

**for (int i = 1; i < infix.size(); i++)**

**{**

**bool isOperator1 = IsOperandNotScobes(infix[i]);**

**bool isOperator2 = IsOperandNotScobes(infix[i-1]);**

**if (isOperator1 && isOperator2)**

**return false;**

**}**

**TStack<char> scobes(infix.size());**

**for (char c : infix) {**

**if (c == '(')**

**{**

**scobes.Push('(');**

**}**

**else if (c == ')') {**

**if (scobes.IsEmpty())**

**{**

**return false;**

**}**

**if (scobes.Top() == '(')**

**{**

**scobes.Pop();**

**}**

**}**

**else if (c == ' ')**

**return false;**

**}**

**return scobes.IsEmpty();**

**}**

**void Expression::ShowPostfix()**

**{**

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

**{**

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

**}**

**cout << endl;**

**}**

**void Expression::InToPostfix()**

**{**

**for (const auto& elem : postfix) {**

**str\_postfix += elem;**

**}**

**}**