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 механик**

ЛАБОРАТОРНАЯ РАБОТА

на тему:

**«РАЗБОР И ВЫЧИСЛЕНИЕ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ С ПОМОЩЬЮ ПОСТФИКСНОЙ ФОРМЫ»**

**Выполнил(а):** студент(ка) группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Суворов Д.И. /

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc154715812)

[1 Постановка задачи 4](#_Toc154715813)

[2 Руководство пользователя 5](#_Toc154715814)

[2.1 Приложение для демонстрации работы стека 5](#_Toc154715815)

[2.2 Приложение для демонстрации перевода и вычисления арифметического выражения 6](#_Toc154715816)

[3 Руководство программиста 8](#_Toc154715817)

[3.1 Алгоритмы 8](#_Toc154715818)

[3.1.1 Стек 8](#_Toc154715819)

[3.1.2 Алгоритм перевода арифметического выражения из инфиксной записи в постфиксную 9](#_Toc154715820)

[3.1.3 Алгоритм вычисления значения арифметического выражения в постфиксной форме 11](#_Toc154715821)

[3.2 Описание классов 12](#_Toc154715822)

[3.2.1 Класс Stack 12](#_Toc154715823)

[3.2.2 Класс ArithmeticExp 13](#_Toc154715824)

[Заключение 16](#_Toc154715825)

[Литературы 17](#_Toc154715826)

[Приложения 18](#_Toc154715827)

[Приложение А. Реализация класса Stack 18](#_Toc154715828)

[Приложение Б. Реализация класса ArithmeticExp 18](#_Toc154715829)

[Приложение В. Использование стека 21](#_Toc154715830)

[Приложение Г. Перевод переданного арифметического выражения и вычисление его значения 22](#_Toc154715831)

# Введение

Арифметическим выражением называется выражение, которое составлено из операндов, соединённых арифметическими операциями, такими как «+», «-», «\*», «/».

Арифметическое выражение может быть записано в различных формах записи. Самые популярные формы: инфиксная, префиксная, постфиксная.

Инфиксная запись – это способ записи, который используем мы в повседневной жизни. Это привычная нам форма записи арифметического выражения, при которой операнды отделяются друг от друга операциями, а порядок действий определяется расстановкой скобок и/или приоритетом операций. Пример выражения в инфиксной записи: (a + b \* c) \* (c / d - e). Но подобная форма записи очень неудобна для автоматического вычисления. Разделение операндов операциями; значимость приоритета операций; наличие круглых скобок, которые могут изменять этот приоритет – всё это делает обработку инфиксного выражения очень сложной задачей.

В 20ых годах прошлого века польским логиком Яном Лукасевичем была разработана префиксная форма записи, в которой были решены эти проблемы. В этой форме записи операция располагается в выражении слева от ее операндов. Таким образом если считать два операнда справа налево, а слева от них стоит операция, то её необходимо выполнить над этими операндами. Круглых скобок нет, приоритет операций больше учитывать не нужно. Для вычисления значения арифметического выражения, записанного в этой форме достаточно один раз пройти по записи справа налево, заранее зная значения каждого операнда. Пример того же выражения, но в инфиксной записи: \* + a \* b c - / c d e.

В 50ых годах прошлого века на основе префиксной записи австралийским ученым Чарльзом Хэмблином была разработана постфиксная форма. В ней операнды располагаются в выражении перед операциями. Плюсы идентичны плюсам префиксной формы, но считывать такую форму для вычисления надо слева направо, что удобнее. Пример постфиксной записи уже описанного выражения: a b c \* + c d / e - \*. Благодаря удобству вычисления выражений, записанных в постфиксной записи, эту запись используют в вычислительной технике и по настоящее время. Для работы с постфиксной формой удобна структура данных стек. Её используют для вычисления результирующего значения выражения в этой форме.

В этой лабораторной работе будет разработан функционал по обработке арифметического выражения в инфиксной форме, переводу его в постфиксную форму и вычисление значения арифметического выражения при заданных значениях каждого операнда. Для описанных

# Постановка задачи

Цель – получить практические навыки применения структуры данных стек для решения задачи о переводе арифметических выражений в постфиксную форму и дальнейшем вычислении его значения.

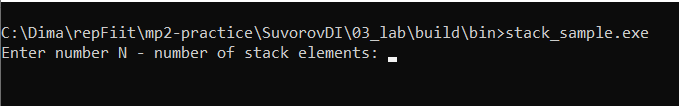
Задачи:

1. Разработать класс, представляющий структуру данных стек.
2. Реализовать все необходимые операции для стека.
3. Разработать класс арифметического выражения.
4. Реализовать возможность перевода арифметического выражения из инфиксной записи с постфиксную.
5. Реализовать возможность вычисления арифметических выражений в постфиксной записи.
6. Протестировать корректность работы разработанного функционала.
7. Сделать вывод о проделанной работе.

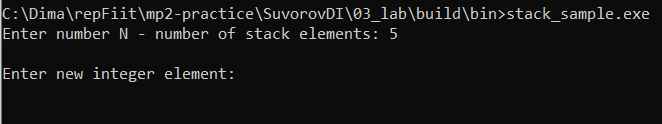
# Руководство пользователя

## Приложение для демонстрации работы стека

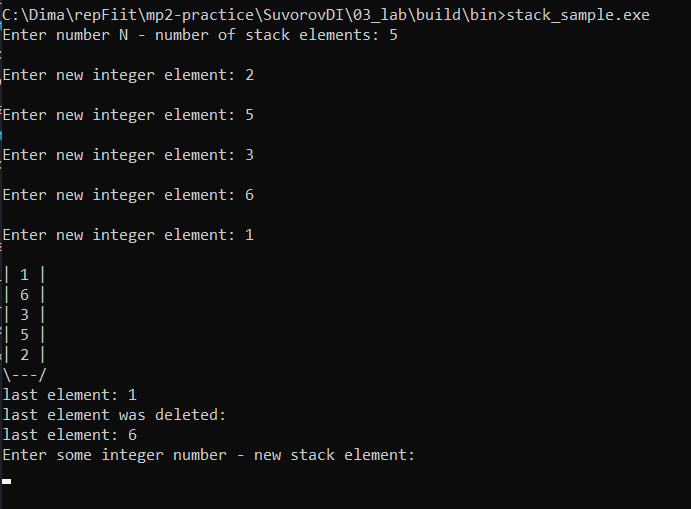
1. Запустить stack\_sample.exe. Сразу после запуска будет выведено следующее   
   (рис. 1).



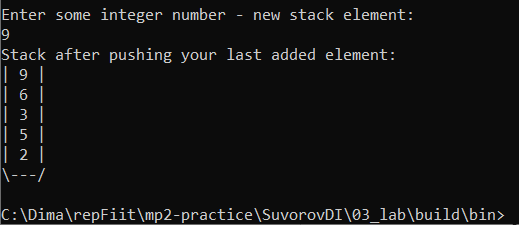
1. Стартовое окно приложения
2. Приложение будет ожидать ввода целочисленного числового значения – количество элементов в стеке. После ввода конкретного значения появится следующее (рис. 2).



1. Состояние приложения после ввода количества элементов стека
2. Приложение будет ожидать ввода целочисленных значений – элементы стека целочисленных значений. После ввода N значений элементов на экране будет выведен стек размера N, элементами которого являются введенные значения (рис. 3).



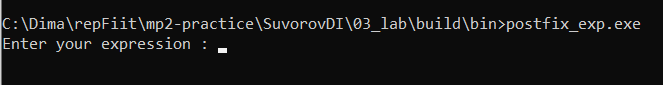
1. Ввод элементов и вывод получившегося на основе них стека
2. На экране также появиться вывод последнего элемента. После этого программа удалит последний элемент из стека и выведет какой элемент стал последним после удаления предыдущего. После этого программа запросит от пользователя любое целочисленное значение – новый элемент, который будет добавлен в стек. После добавления нового элемента изменённый стек будет выведен и приложение закончит свою работу (рис. 4).



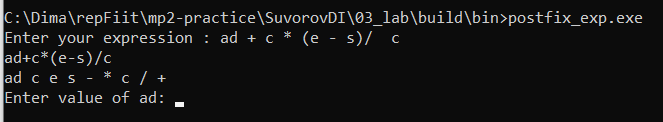
1. Итоговое окно приложения

## Приложение для демонстрации перевода и вычисления арифметического выражения

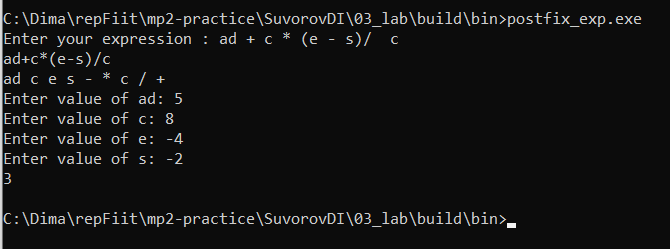
1. Запустить postfix\_exp.exe. В результате появится следующее окно (рис. 5).



1. Стартовое окно приложения
2. Приложение ожидает ввода изначального выражения в инфиксной форме. После ввода выражения приложение выведет его в инфиксной форме и в постфиксной (рис. 6).



1. Ввод арифметического выражения и результат его перевода в постфиксную форму
2. После вывода форм выражения приложение будет ожидать ввода значений каждого операнда для вычисления значения арифметического выражения   
   (рис. 7).



1. Итоговое окно приложения
2. В результате приложение выведет результат вычисления. Приложение после этого заканчивает свою работу.

# Руководство программиста

## Алгоритмы

### Стек

Стек – это динамическая структура данных построенная по принципу «последним вошел – первым вышел» (LIFO – Last In First Out). У стека есть три основные операции: Push, Pop, Top. Push – операция добавления элемента в стек; Pop – это операция удаления элемента из стека; Top – это операция показывает элемент с вершины стека. Все эти операции выполняют своё действия с элементом, находящимся на вершине стека (т.е. с элементом, который был последним добавлен в стек). Эти операции полностью характеризуют и определяют функциональные возможности стека, а именно:

1) В каждый момент времени доступен только элемент с вершины стека с помощью операции Top.

2) Операция добавления элемента – добавляет элемент на вершину стека. Добавленный элемент устанавливается на вершину и соответственно становится доступным.

3) Операция удаления – удаляет элемент с вершины стека. Элементом на вершине стека становится элемент, который находился в стеке сразу после удалённого элемента.

Принцип работы стека можно сравнить со стопкой тарелок – взять и положить новую тарелку можно только сверху стопки.

Подобное поведение структуры данных стек можно визуализировать:

Пусть дан стек, состоящий из 5 элементов:

|  |  |
| --- | --- |
| Top → | **3** |
|  | -5 |
|  | 6 |
|  | -2 |
|  | 4 |

Применив к этому стеку операцию Top в результате мы получим значение 3 – это элемент с вершины стека.

Применив операцию Push со значением нового элемента 1 в результате на вершину стека установится 1:

|  |  |
| --- | --- |
| Top → | **1** |
|  | 3 |
|  | -5 |
|  | 6 |
|  | -2 |
|  | 4 |

Теперь элементом на вершине стека является 1 и любая из операций будут воздействовать на нее. Например, применим к стеку операцию Pop. Тогда произойдет удаление вершинного элемента из стека. На вершине окажется элемент 3, который находился после удалённого элемента 1:

|  |  |
| --- | --- |
| Top → | **3** |
|  | -5 |
|  | 6 |
|  | -2 |
|  | 4 |

Снова элемент 3 находится на вершине стека.

Также для стека определена еще одна операция – проверка стека на пустоту. Эту операция будет возвращать true, если стек пуст (т.е. в нём нет элементов) и false, если в стеке будут элементы.

### Алгоритм перевода арифметического выражения из инфиксной записи в постфиксную

На вход программе подается строка – арифметическое выражение в инфиксной форме. На выход программа выдает строку – арифметическое выражение в постфиксной форме.

Для того, чтобы перевести запись арифметического выражения из инфиксной формы в постфиксную будет использована динамическая структура данных стек.

Подготовительный этап:

Использоваться будут 2 стека. Один будет предназначен для операндов, а другой для операций. Доступный список операций: «+», «-», «\*», «/», «(», «)». Для первых четырёх операций вводится приоритет, который будет задаваться числовым значением от 1 до 3: «\*», «/» – 3; «+», «-» – 2; «(» – 1.

Алгоритм перевода:

Арифметическое выражение в инфиксной форме просматривается слева направо:

1) Если встречается операнд – он добавляется на вершину стека операндов.

2) Если встречается операция «(», то она добавляется на вершину стека операций.

3) Если встречается «)», то из стека операций последовательно изымаются элементы в стек операндов, пока не будет изъята операция «(» или пока стек операций не станет пустым. (Изымается, значит на вершину стека операндов добавляется элемент, находящийся на вершине стека операций. После этого изъятый элемент удаляется с вершины стека операций).

4) Если встречена операция, которая обладает приоритетом, то её приоритет сравнивается с приоритетом операции, находящейся в данный момент на вершине стека операций. Пока приоритет новой операции численно более низкий, чем на вершине стека операций, операции на вершине стека будут последовательно изыматься в стек операндов (процесс изъятия идентичен описанному в предыдущем пункте). После изъятия всех подобных операций – новая добавляется на вершину стека операций. При этом операция «(» не изымается из стека операций в стек операндов, а просто удаляется.

После того, как инфиксная форма полностью просмотрено из стека операций последовательно изымаются все операции в стек операндов, пока стек операций не станет пустым.

Итоговая постфиксная запись будет храниться в стеке операндов.

Пример применения алгоритма перевода арифметического выражения в постфиксную запись:

Исходное арифметическое выражение в инфиксной записи: a + (b - c) \* d – f / (g + h).

Цифрой **1** обозначен стек операндов. Цифрой **2** обозначен стек операций.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | - |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | / |
|  |  |  |  |  |  |  |  |  |  |  |  |  | + | + |
|  |  |  |  |  |  |  |  |  |  |  |  | h | h | h |
|  |  |  |  |  |  |  |  |  |  |  | g | g | g | g |
|  |  |  |  |  |  |  |  |  | f | f | f | f | f | f |
|  |  |  |  |  |  |  |  | + | + | + | + | + | + | + |
|  |  |  |  |  |  |  |  | \* | \* | \* | \* | \* | \* | \* |
|  |  |  |  |  |  |  | d | d | d | d | d | d | d | d |
|  |  |  |  |  | - | - | - | - | - | - | - | - | - | - |
|  |  |  |  | c | c | c | c | c | c | c | c | c | c | c |
|  |  | b | b | b | b | b | b | b | b | b | b | b | b | b |
| a | a | a | a | a | a | a | a | a | a | a | a | a | a | a |
| 2 |  |  |  |  |  |  |  |  |  |  |  | + | + |  |  |
|  |  |  | - | - |  |  |  |  |  | ( | ( | ( |  |  |
|  | ( | ( | ( | ( |  | \* | \* |  |  | / | / | / | / |  |
| + | + | + | + | + | + | + | + | - | - | - | - | - | - |  |

**Итоговая постфиксная запись**: a b c – d \* + f g h + / -.

### Алгоритм вычисления значения арифметического выражения в постфиксной форме

На вход этому алгоритму подается запись постфиксной формы выражения и массив значений всех операндов, использованных в этом выражении. На выход число – результат вычисления значения переданного арифметического выражения.

Для вычисления будет необходим стек.

Алгоритм вычисления:

Последовательно считывает по 1 символу из постфиксной записи пока не будет достигнут конец записи. Для каждого символа выполняем:

1) Если прочитан операнд, то добавляет его значение на вершину стека.

2) Если прочитан знак операции, то считываем два верхних значения из стека и выполняем для этих операндов операцию, которую обозначает прочитанный знак. Элемент считанный с вершины стека последним – будет левым операндом, а считанный первым – правым.

Как только будет достигнут конец входной строки – на вершине стека будет находится число, являющееся результатом вычисления арифметического выражения.

Пример вычисления значения арифметического выражения:

Исходное арифметическое выражение в инфиксной форме: a + (b - c) \* d – f / (g + h).

Постфиксная форма этого выражения: a b c – d \* + f g h + / -.

На вход подается массив значений всех операндов арифметического выражения, значение которого необходимо вычислить:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | f | g | h |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Цифрой 1 обозначен стек, в котором будет происходить вычисление:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  | 7 |  |  |  |
|  |  | 3 |  | 4 |  |  | 6 | 13 |  |  |
|  | 2 | 2 | -1 | -1 | -4 |  | 5 | 5 | 2,6 |  |
| 1 | 1 | 1 | 1 | 1 | 1 | -3 | -3 | -3 | -3 | **-5,6** |

Результатом вычисления заданного арифметического выражения при заданных значениях операндов равно **6,5**.

## Описание классов

### Класс Stack

Объявление класса:

template <typename ValueType>

class Stack {

private:

  int top;

  std::vector<ValueType> stack;

public:

  Stack();

  Stack(const Stack<ValueType>& s);

  bool operator==(const Stack<ValueType>& st);

  void Pop();

  ValueType Top() const;

  void Push(const ValueType& elem);

  bool isEmpty() const;

  friend std::ostream& operator<<(std::ostream& out, Stack<ValueType>& s);

};

Поля:

top – индекс элемента на вершине стека.

stack – память для хранения элементов стека.

Методы:

bool operator==(const Stack<ValueType>& st);

Назначение: сравнение на равенство двух стеков.

Входные данные: ссылка на константный экземпляр стека, с которым необходимо сравнить текущий стек.

Выходные данные: логическое значение true, если стеки равны; false в противном случае.

void Pop();

Назначение: удаление элемента с вершины стека.

ValueType Top() const;

Назначение: получение значения элемента с вершины стека.

Выходные данные: значение элемента на вершине стека.

bool isEmpty() const;

Назначение: проверка стека на пустоту.

Выходные данные: логическое значение true, если стек пуст (т.е. в нем нет элементов); false в противном случае.

friend std::ostream& operator<<(std::ostream& out, Stack<ValueType>& s);

Назначение: вывод элементов стека.

Входные данные: ссылка на поток вывода и ссылка на стек, элементы которого необходимо вывести.

Выходные данные: ссылка на поток вывода.

### Класс ArithmeticExp

class ArithmeticExp {

private:

  std::string original\_exp;

  std::vector<std::string> vector\_postfix\_exp;

  std::string string\_postfix\_exp;

  std::vector<std::string> lexemes;

  std::map<std::string, int> priorities;

  std::map<std::string, double> operands\_values;

  std::vector<std::string> valid\_symbols;

  void Parse();

  void ToPostfix();

  bool Check();

public:

  ArithmeticExp(const std::string &exp);

  std::string GetOriginal() const;

  std::string GetPostfix() const;

  std::vector<std::string> GetOperands() const;

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

};

Поля:

original\_exp – инфиксная форма арифметического выражения в виде строки.

vector\_postfix\_exp – инфиксная форма арифметического выражения в виде вектора.

string\_postfix\_exp – инфиксная форма арифметического выражения в виде строки.

lexemes – все символы инфиксной формы арифметического выражения (символы операндов и знаки операций).

priorities – приоритеты каждой операции.

operands\_values – значения всех операндов арифметического выражения.

valid\_symbols – допустимые символы в строке арифметического выражения.

Методы:

void Parse();

Назначение: получает список имен всех операндов и знаков всех операций, использованных в инфиксной форме арифметического выражения.

void ToPostfix();

Назначение: перевод формы арифметического выражения из инфиксной в постфиксную.

bool Check();

Назначение: проверка инфиксной формы арифметического выражения на корректность и возможность перевода.

Выходные данные: логическое значение true, если исходное арифметическое выражение корректно, false – в обратном случае.

std::string GetOriginal() const;

Назначение: получение исходной инфиксной формы арифметического выражения.

Выходные данные: строка исходной инфиксной формы арифметического выражения.

std::string GetPostfix() const;

Назначение: получение итоговой постфиксной формы арифметического выражения.

Выходные данные: строка постфиксной формы арифметического выражения.

std::vector<std::string> GetOperands() const;

Назначение: получение списка имен всех операндов, использованных в арифметическом выражении.

Выходные данные: вектор, элементами которого являются имена всех операндов выражения.

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

Назначение: вычисление значения арифметического выражения в постфиксной записи.

Входные данные: значения всех операндов арифметического выражения.

Выходные данные: результат вычисления значения.

# Заключение

В результате данной лабораторной работы были получены практические навыки применения динамической структуры данных стек. На основе полученных знаний в этой предметной области была разработана программа, которая реализует обработку арифметических выражений в инфиксной форме, перевод их в постфиксную форму и вычисление значения выражения в постфиксной форме. Была произведена проверка корректной работы реализованного функционала с различными наборами данных. В целом, лабораторная работа помогла понять основные принципы работы со стеком.

# Литературы

1. Стек, теоретические основы [https://cloud.unn.ru/s/6Cjm3qYzBsZxHi6]
2. Теоретические основы форм записи арифметических выражений и алгоритмы перевода и вычисления значения арифметического выражения в постфиксной записи [https://cloud.unn.ru/s/4Pyf24EBmowGsQ2]

# Приложения

## Приложение А. Реализация класса Stack

template <typename ValueType>

class Stack {

private:

  int top;

  std::vector<ValueType> stack;

public:

  Stack() : top(-1) {}

  Stack(const Stack<ValueType>& s) : top(s.top), stack(s.stack) {}

  bool operator==(const Stack<ValueType>& st) {

      return (top == st.top) && (stack == st.stack);

  }

  void Pop() {

    if (isEmpty())

      throw std::exception("The stack is empty\n");

    stack.pop\_back();

    top--;

  }

  ValueType Top() const {

    if (isEmpty())

      throw std::exception("The stack is empty\n");

    return stack[top];

  }

  void Push(const ValueType& elem) {

    stack.push\_back(elem);

    top++;

  }

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

  friend std::ostream& operator<<(std::ostream& out, Stack<ValueType>& s) {

      Stack<ValueType> tmp = s;

      while (!s.isEmpty()) {

        std::cout << "| " << s.Top() << " |" << std::endl;

        s.Pop();

      }

      std::cout << "\\---/" << std::endl;

      s = tmp;

      return out;

  }

};

## Приложение Б. Реализация класса ArithmeticExp

bool isLetter(char l) {

  return ('a' <= l && l <= 'z') || ('A' <= l && l <= 'Z');

}

bool valid\_brackets(const std::string& str)

{

  int count = 0;

  for (char c : str)

  {

    if (c == '(')

      count++;

    else if (c == ')')

      count--;

    if (count < 0)

      return false;

  }

  return count == 0;

}

bool is\_char\_operation(char c, std::vector<std::string> valid\_symbols) {

  return std::find(valid\_symbols.begin(), valid\_symbols.end(), std::string(1, c)) != valid\_symbols.end();

}

bool ArithmeticExp::Check() {

  for (const char i : original\_exp) {

    if (!(isLetter(i) || is\_char\_operation(i, valid\_symbols))) {

      return false;

    }

  }

  if (!valid\_brackets(original\_exp))

    return false;

  if (original\_exp == "")

    return false;

  return true;

}

void ArithmeticExp::Parse() {

  std::string name = "";

  for (char c : original\_exp) {

    if (!is\_char\_operation(c, valid\_symbols)) {

      name += c;

    }

    else {

      if (name != "") {

        lexemes.push\_back(name);

      }

      name = "";

      lexemes.push\_back(std::string(1, c));

    }

  }

  if (name != "") {

    lexemes.push\_back(name);

  }

}

ArithmeticExp::ArithmeticExp(const std::string& exp) : original\_exp(exp) {

  original\_exp.erase(std::remove\_if(original\_exp.begin(), original\_exp.end(), isspace), original\_exp.end());

  priorities = {

    {"\*", 3},

    {"/", 3},

    {"+", 2},

    {"-", 2},

    {"(", 1},

  };

  valid\_symbols = {

    ")",

  };

  for (std::pair<std::string, int> i : priorities) {

    valid\_symbols.push\_back(i.first);

  }

  bool check\_status = Check();

  if (check\_status) {

    ToPostfix();

  }

  else {

    throw std::exception("INVALID EXPRESSION");

  }

}

void ArithmeticExp::ToPostfix() {

  Parse();

  Stack<std::string> operations;

  Stack<std::string> operands;

  for (std::string elem : lexemes) {

    switch (elem[0]) {

    case '(':

      operations.Push(elem);

      break;

    case ')':

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

        operands.Push(operations.Top());

        operations.Pop();

      }

      operations.Pop(); // deleting '('

      break;

    case '+': case '-': case '\*': case '/':

      while (!operations.isEmpty()) {

        if (priorities[elem] <= priorities[operations.Top()]) {

          operands.Push(operations.Top());

          operations.Pop();

        }

        else {

          break;

        }

      }

      operations.Push(elem);

      break;

    default:

      operands.Push(elem);

      operands\_values.insert({ elem, 0.0 });

    }

  }

  while (!operations.isEmpty()) {

    operands.Push(operations.Top());

    operations.Pop();

  }

  Stack<std::string> tmp(operands);

  while (!tmp.isEmpty()) {

    vector\_postfix\_exp.insert(vector\_postfix\_exp.begin(), tmp.Top());

    tmp.Pop();

  }

  while (!operands.isEmpty()) {

    string\_postfix\_exp = operands.Top() + " " + string\_postfix\_exp;

    operands.Pop();

  }

}

std::vector<std::string> ArithmeticExp::GetOperands() const {

  std::vector<std::string> op;

  for (const std::pair<std::string, double> elem : operands\_values) {

    op.push\_back(elem.first);

  }

  return op;

}

double ArithmeticExp::Calculate(const std::map<std::string, double>& values) {

  for (std::pair<std::string, double> val : values) {

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

  }

  Stack<double> st;

  double leftOp, rightOp;

  for (std::string lexem : vector\_postfix\_exp) {

    if (lexem == "+") {

      rightOp = st.Top();

      st.Pop();

      leftOp = st.Top();

      st.Pop();

      st.Push(leftOp + rightOp);

    }

    else if (lexem == "-") {

      rightOp = st.Top();

      st.Pop();

      leftOp = st.Top();

      st.Pop();

      st.Push(leftOp - rightOp);

    }

    else if (lexem == "\*") {

      rightOp = st.Top();

      st.Pop();

      leftOp = st.Top();

      st.Pop();

      st.Push(leftOp \* rightOp);

    }

    else if (lexem == "/") {

      rightOp = st.Top();

      st.Pop();

      leftOp = st.Top();

      st.Pop();

      if (rightOp == 0.0) {

        throw std::exception("ZERO\_DIVISION");

      }

      st.Push(leftOp / rightOp);

    }

    else {

      st.Push(operands\_values[lexem]);

    }

  }

  return st.Top();

}

## Приложение В. Использование стека

#include "stack.h"

#include <iostream>

using namespace std;

int main() {

    Stack<int> a;

    int n;

    cout << "Enter number N - number of stack elements: ";

    cin >> n;

    cout << endl;

    for (int i = 0; i < n; i++) {

        int el;

        cout << "Enter new integer element: ";

        cin >> el;

        cout << endl;

        a.Push(el);

    }

    cout << a;

    cout << "last element: " << a.Top() << endl;

    a.Pop();

    if (!a.isEmpty()) {

      cout << "last element was deleted:" << endl;

      cout << "last element: " << a.Top() << endl;

    }

    else {

      cout << "Your stack is empty now!" << endl;

    }

    int new\_element;

    cout << "Enter some integer number - new stack element: " << endl;

    cin >> new\_element;

    a.Push(new\_element);

    cout << "Stack after pushing your last added element:" << endl;

    cout << a;

    return 0;

}

## Приложение Г. Перевод переданного арифметического выражения и вычисление его значения

#include "postfix\_exp.h"

#include <iostream>

#include <string>

#include <vector>

#include <map>

int main() {

    std::string exp;

    std::cout << "Enter your expression : ";

    std::getline(std::cin, exp);

    ArithmeticExp expr(exp);

    std::cout << expr.GetOriginal() << std::endl;

    std::cout << expr.GetPostfix() << std::endl;

    if (expr.GetPostfix() != "there are invalid characters...") {

        std::vector<std::string> operands = expr.GetOperands();

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

        double val;

        for (const std::string op : operands) {

            std::cout << "Enter value of " << op << ": ";

            std::cin >> val;

            values[op] = val;

        }

        std::cout << expr.Calculate(values) << std::endl;

    }

    return 0;

}