МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
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**

Направление подготовки: «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе

**Вычисление арифметических выражений (стеки)**

Выполнил:

студент группы 3822Б1ПР2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ермолаев В.А.

Подпись

Нижний Новгород

2023 г.

Содержание

[1. Введение 3](#_Toc153920479)

[2. Постановка задачи 4](#_Toc153920480)

[3. Руководство пользователя 5](#_Toc153920481)

[4. Руководство программиста 7](#_Toc153920482)

[4.1. Описание структуры программы 7](#_Toc153920483)

[4.2. Описание структур данных 7](#_Toc153920484)

[4.2.1. TStack (шаблонный класс) 7](#_Toc153920485)

[4.2.2. TArithmeticExpression 7](#_Toc153920486)

[5. Эксперименты 9](#_Toc153920487)

[6. Заключение 10](#_Toc153920488)

[7. Литература 11](#_Toc153920489)

[8. Приложение 1 12](#_Toc153920490)

[9. Приложение 2 13](#_Toc153920491)

[10. Приложение 3 14](#_Toc153920492)

[11. Приложение 4 16](#_Toc153920493)

[12. Приложение 5 18](#_Toc153920494)

[13. Приложение 6 19](#_Toc153920495)

[14. Приложение 7 23](#_Toc153920496)

[15. Приложение 8 24](#_Toc153920497)

[16. Приложение 9 27](#_Toc153920498)

[17. Приложение 10 28](#_Toc153920499)

[18. Приложение 11 30](#_Toc153920500)

[19. Приложение 12 31](#_Toc153920501)

[20. Приложение 13 34](#_Toc153920502)

[21. Приложение 14 36](#_Toc153920503)

1. Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Стек. С этой целью в лабораторной работе изучаются различные варианты структуры хранения стеков и разрабатываются методы и программы решения ряда задач с использованием стеков. В качестве области приложений выбрана тема вычисления арифметических выражений, возникающей при трансляции программ на языке программирования высокого уровня в исполняемые программы.

При вычислении произвольных арифметических выражений возникают две основные задачи: проверка корректности введённого выражения и выполнение операций в порядке, определяемом их приоритетами и расстановкой скобок. Существует алгоритм, позволяющий реализовать вычисление произвольного арифметического выражения за один просмотр без хранения промежуточных результатов. Для реализации данного алгоритма выражение должно быть представлено в постфиксной форме. Рассматриваемые в данной лабораторной работе алгоритмы являются начальным введением в область машинных вычислений

1. Постановка задачи

Арифметическое выражение - выражение, в котором операндами являются объекты, над которыми выполняются арифметические операции. Например, (1+2)/(3+4\*6.7)-5.3\*4.4

При такой форме записи (называемой инфиксной, где знаки операций стоят между операндами) порядок действий определяется расстановкой скобок и приоритетом операций. Постфиксная (или обратная польская) форма записи не содержит скобок, а знаки операций следуют после соответствующих операндов. Тогда для приведённого примера постфиксная форма будет иметь вид: 1 2+ 3 4 6.7\*+/ 5.3 4.4\* -

Обратная польская нотация была разработана австралийским ученым Чарльзом Хэмблином в середине 50-х годов прошлого столетия на основе польской нотации, которая была предложена в 1920 году польским математиком Яном Лукасевичем. Эта нотация лежит в основе организации вычислений для арифметических выражений. Известный ученый Эдсгер Дейкстра предложил алгоритм для перевода выражений из инфиксной в постфиксную форму. Данный алгоритм основан на использовании стека.

Стек (англ. stack), магазин – схема запоминания информации, при которой каждый вновь поступающий ее элемент как бы «проталкивает» вглубь отведенного участка памяти находящиеся там элементы (подобно патрону, помещаемому в магазин винтовки) и занимает крайнее положение (так называемую вершину стека). При выдаче информации из стека выдается элемент, расположенный в вершине стека, а оставшиеся элементы продвигаются к вершине; следовательно, элемент, поступивший последним, выдается первым [1]. Более строгое определение структуры дано в разделе 2 описания данной лабораторной работы.

1. Руководство пользователя

При запуске программы необходимо ввести два полинома в текстовые первого и второго блока соответствующе (также присутствуют полиномы по умолчанию). После чего выбрать операцию над введенными полиномами.

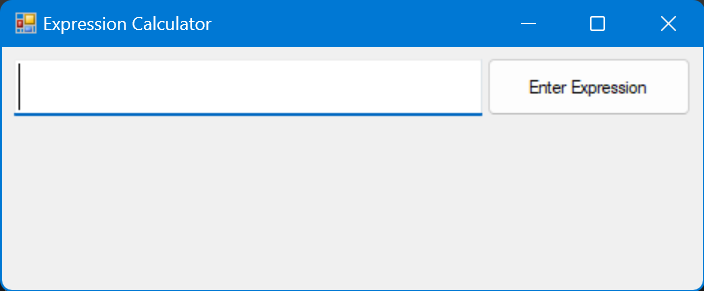


Рис. 1 – Исходное состояние

После выбора операции будет выведен результат.

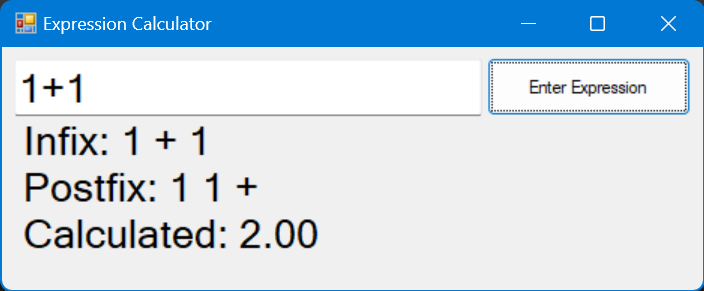


Рис. 2 – Результат

Если в выражении есть переменные, то будет вызванно окно ввода

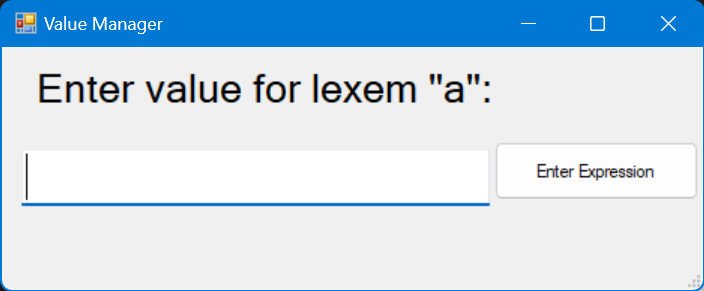


Рис. 3 – Ввод значения переменной

В случае возникновения ошибки во время исполнения, будет вызвано окно-сообщения с ошибкой

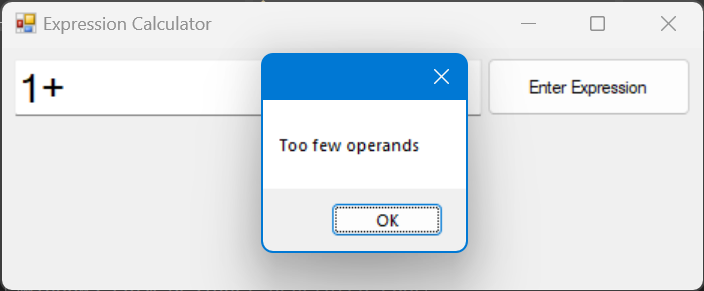


Рис. 5 – Сообщение об ошибке

1. Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль Expression, содержащий реализацию класса TArithmeticExpression (файлы в директориях ./include/ и ./src/).
* Модуль Expression\_test. Набор тестов для класса TAritthmeticExpression. Включает в себя файл ./test/test\_expressuib.cpp. Реализованы они с помощью использования фреймворка Google Test.
* Примеры использования классов (файлы в директории ./samples/).
* Графическое приложение Expression\_gui (файлы в директории ./gui/)

## Описание структур данных

### TStack (шаблонный класс)

protected:

size\_t size – размер стека

size\_t top – количество элементов

T\* mas – массив значений

public:

TStack(size\_t size = 1) – конструктор

TStack(TStack<T>& stack) noexcept – конструктор копирования

~TStack() noexcept - деструктор

void Push(T element) – положить элемент в стек

void Pop() – удалить элемент из стека

T TopView() – посмореть верхний элемент

size\_t GetSize() noexcept – получить размер стека

size\_t GetTop() noexcept – получить количество элементов

bool IsFull() noexcept – проверка на заполненность

bool IsEmpty() noexcept – проверка на опустошенность

TStack<T>& operator=(const TStack<T>& stack) noexcept;

friend std::ostream& operator<<(std::ostream& ostr, const TStack<T>& stack) noexcept – оператор вывода

friend std::istream& operator>>(std::istream& istr, const TStack<T>& stack) noexcept – оператор ввода

### TArithmeticExpression

static const std::map<std::string, int> priority – поле хеш-таблица «операция-приоритет»

std::string infix – поле инфиксная запись

std::string postfix – поле постфиксная запись

std::map<std::string, double> operands – поле хэш-таблица «переменная-значение»

std::vector<std::string> Parse(const std::string& inp, TypeParse type) – разбивает строку на операнды и операторы.

void ToPostfix() – приводит выражение к постфиксному виду, сохраняет результат в поле postfix

void ToInfix() - приводит выражение к инфиксному виду, сохраняет результат в поле infix

void IsValidExpression(); - проверка корректности выражения

public:

TArithmeticExpression(std::string& infix) - конструктор

std::string GetInfix() const – получить инфиксную запись

std::string GetPostfix() const – получить постфиксную запись

std::vector<std::string> GetOperands() const – получить имена операндов

double Calculate(const std::map < std::string , double > & values) – вычислить постфиксную запись

- перечисление констант для запуска тестов производительности

1. Эксперименты

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 11
2. Процессор: Intel(R) Core™ i3-1115G4- CPU @ 3.00 GHz
3. Версия Visual Studio: 2022

Эксперименты показали, что добавление и изъятие из очереди происходят за константое время, т.е. сложность алгоримов – .

1. Заключение

В заключение, данная лабораторная работа позволяет освоить работу с динамической структурой данных Стек. В ходе работы изучаются различные варианты структуры хранения стеков и разрабатываются методы и программы для решения задач с использованием стеков. Основное направление работы - вычисление арифметических выражений, особенно при трансляции программ на языке программирования высокого уровня в исполняемые программы.

Для решения задач вычисления произвольных арифметических выражений существует алгоритм, позволяющий выполнить вычисление за один просмотр без хранения промежуточных результатов. Для этого выражение должно быть представлено в постфиксной форме. Разработанные в данной лабораторной работе алгоритмы представляют собой введение в область машинных вычислений.

Таким образом, лабораторная работа позволяет практически освоить структуру данных Стек и получить начальные навыки вычисления арифметических выражений в постфиксной форме. Эти навыки могут быть полезными при разработке программных решений в будущем.

1. Литература
   * + 1. Лабораторный практикум. Составители: Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

1. Приложение 1

MyStack.h

#ifndef \_MY\_STACK\_

#define \_MY\_STACK\_

#include <iostream>

template <class T>

class TStack {

protected:

    size\_t size;

    size\_t top;

    T\* mas;

public:

    TStack(size\_t size = 1);

    TStack(TStack<T>& stack) noexcept;

    ~TStack() noexcept;

    void Push(T element);

    void Pop();

    T TopView();

    size\_t GetSize() noexcept;

    size\_t GetTop() noexcept;

    bool IsFull() noexcept;

    bool IsEmpty() noexcept;

    TStack<T>& operator=(const TStack<T>& stack) noexcept;

    //операторы вводы и выводы

    friend std::ostream& operator<<(std::ostream& ostr, const TStack<T>& stack) noexcept{

        for (size\_t i = 0; i < stack.size; i++) {

            ostr << i + 1 << ". " << stack.mas[i] << std::endl;

        }

        return ostr;

    }

    friend std::istream& operator>>(std::istream& istr, const TStack<T>& stack) noexcept{

        for (size\_t i = 0; i < stack.size; i++) {

            istr >> stack.mas[i];

        }

        return istr;

    }

};

#include "../src/MyStack.hpp"

#endif

1. Приложение 2

stack\_sample.h

#include <iostream>

#include "MyStack.h"

using namespace std;

int main(int argc, char\*\* argv) {

    setlocale(LC\_ALL, "Russian");

    size\_t size, op;

    int el;

    try {

        cout << "Введите размер стека: ";

        cin >> size;

        TStack<int> s(size);

        while (true) {

            cout << "Меню операций:\n"

                << "1. Добавить в стек\n"

                << "2. Удалить элемент\n"

                << "3. Верхний элемент\n"

                << "4. Количество элементов\n"

                << "5. Размер\n"

                << "Ввести операцию: ";

            cin >> op;

            cout << endl;

            switch (op) {

            case 1:

                cout << "Введите элемент (int): ";

                cin >> el;

                s.Push(el);

                break;

            case 2:

                s.Pop();

                break;

            case 3:

                cout << s.TopView() << endl;

                break;

            case 4:

                cout << s.GetTop() << endl;

                break;

            case 5:

                cout << s.GetSize() << endl;

                break;

            default:

                break;

            }

            cout << s << endl;

        }

    }

    catch (exception e) {

        cerr << e.what() << endl;

    }

}

1. Приложение 3

MyStack.hpp

#include "MyStack.h"

template<class T>

TStack<T>::TStack(size\_t size)

{

    if (size == 0)

        throw std::domain\_error("Stack size = 0 throws Heap Curruption Error");

    this->size = size;

    this->top = 0;

    this->mas = new T[this->size]();

}

template<class T>

TStack<T>::TStack(TStack<T>& stack) noexcept

{

    this->size = stack.size;

    this->top = stack.top;

    this->mas = new T[this->size]();

    std::copy(stack.mas, stack.mas + stack.size, this->mas);

}

template<class T>

TStack<T>::~TStack() noexcept

{

    if (this->mas != nullptr) {

        delete[] this->mas;

        this->mas = nullptr;

    }

    this->size = 0;

    this->top = 0;

}

template<class T>

void TStack<T>::Push(T element)

{

    if (this->IsFull())

        throw std::logic\_error("Stack overflow");

    this->mas[this->top++] = element;

}

template<class T>

void TStack<T>::Pop()

{

    if (this->IsEmpty())

        throw std::logic\_error("Stack is empty");

    this->mas[--this->top] = T();

}

template<class T>

T TStack<T>::TopView()

{

    if (this->IsEmpty())

        throw std::logic\_error("Stack is empty");

    return this->mas[this->top - 1];

}

template<class T>

size\_t TStack<T>::GetSize() noexcept

{

    return this->size;

}

template<class T>

size\_t TStack<T>::GetTop() noexcept

{

    return this->top;

}

template<class T>

bool TStack<T>::IsFull() noexcept

{

    return this->top == this->size;

}

template<class T>

bool TStack<T>::IsEmpty() noexcept

{

    return this->top == 0;

}

template<class T>

TStack<T>& TStack<T>::operator=(const TStack<T>& stack) noexcept

{

    if (this == &stack)

        return \*this;

    if (this->size != stack.size) {

        if (this->mas != nullptr)

            delete[] this->mas;

        this->size = stack.size;

        this->mas = new T[this->size]();

    }

    this->top = stack.top;

    std::copy(stack.mas, stack.mas + stack.size, this->mas);

    return \*this;

}

1. Приложение 4

test\_stack.h

#include "MyStack.h"

#include <gtest.h>

TEST(TStack, cant\_create\_with\_zero\_size)

{

    ASSERT\_ANY\_THROW(TStack<int> s1(0));

}

TEST(TStack, can\_create)

{

    ASSERT\_NO\_THROW(TStack<int> s1(10));

}

TEST(TStack, can\_get\_size)

{

    TStack<int> s1(2);

    ASSERT\_EQ(2, s1.GetSize());

}

TEST(TStack, can\_get\_top)

{

    TStack<int> s1(2);

    ASSERT\_EQ(0, s1.GetTop());

}

TEST(TStack, can\_push\_element)

{

    TStack<int> s1(1);

    ASSERT\_NO\_THROW(s1.Push(10));

}

TEST(TStack, cant\_push\_element\_when\_stack\_is\_full)

{

    TStack<int> s1(1);

    s1.Push(10);

    ASSERT\_ANY\_THROW(s1.Push(10));

}

TEST(TStack, can\_pop\_element)

{

    TStack<int> s1(1);

    s1.Push(10);

    ASSERT\_NO\_THROW(s1.Pop());

}

TEST(TStack, cant\_pop\_element\_when\_stack\_is\_empty)

{

    TStack<int> s1(1);

    ASSERT\_ANY\_THROW(s1.Pop());

}

TEST(TStack, can\_get\_top\_element)

{

    TStack<int> s1(1);

    s1.Push(10);

    ASSERT\_EQ(10, s1.TopView());

}

TEST(TStack, cant\_get\_top\_element\_when\_stack\_is\_empty)

{

    TStack<int> s1(1);

    ASSERT\_ANY\_THROW(s1.TopView());

}

TEST(TStack, can\_check\_is\_empty)

{

    TStack<int> s1(2);

    ASSERT\_EQ(true, s1.IsEmpty());

    s1.Push(1);

    ASSERT\_EQ(false, s1.IsEmpty());

}

TEST(TStack, can\_check\_is\_full)

{

    TStack<int> s1(1);

    ASSERT\_EQ(false, s1.IsFull());

    s1.Push(1);

    ASSERT\_EQ(true, s1.IsFull());

}

TEST(TStack, can\_assign\_stack\_to\_itself)

{

    TStack<int> s1(2);

    ASSERT\_NO\_THROW(s1 = s1);

}

TEST(TStack, can\_assign\_stacks\_with\_different\_sizes)

{

    TStack<int> s1(2), s2(3);

    ASSERT\_NO\_THROW(s1 = s2);

}

1. Приложение 5

MyExpression.h

#ifndef \_MY\_EXPRESSION\_

#define \_MY\_EXPRESSION\_

#define PI 3.14

#include "../../stack/include/MyStack.h"

#include <string>

#include <sstream>

#include <vector>

#include <map>

#include <cmath>

#include <regex>

#include <format>

enum TypeParse {

    INFIX,

    POSTFIX

};

/\*

    \* Класс реализует базовые функции:

        – Получение, хранение и возврат исходной (инфиксной) формы

        – Преобразование в постфиксную форму

        – Возврат постфиксной формы

        – Вычисление арифметического выражения при заданных значениях операндов

    \* Добавленные раширения функциональности:

        – Поддержа неодносимвольных имен операндов

        – Поддержка констант в выражении, как целых, так и вещественных (а также константа PI)

        – Поддержка унарных префиксных (например, унарный минус) и постфиксных (например, факториал) операций

        – Поддержка функций

            - sin

            - cos

            - tg

            - ctg

        – Возможность использовать разные виды скобок: (), {}, []

        - Анализ корректности выражения, c выдачей позиции и описания ошибки

    \*/

class TArithmeticExpression

{

    static const std::map<std::string, int> priority;

    std::string infix;

    std::string postfix;

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

    std::vector<std::string> Parse(const std::string& inp, TypeParse type);

    void ToPostfix();

    void ToInfix();

    void IsValidExpression();

public:

    TArithmeticExpression(std::string& infix);

    std::string GetInfix() const { return std::regex\_replace(infix.data(), std::regex("~"), "-"); }

    std::string GetPostfix() const { return postfix; }

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

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

};

#endif

1. Приложение 6

MyExpression.cpp

#include "MyExpression.h"

TArithmeticExpression::TArithmeticExpression(std::string& infix) : infix(infix)

{

    IsValidExpression();

    ToInfix();

    ToPostfix();

}

std::vector<std::string> TArithmeticExpression::Parse(const std::string& inp, TypeParse type)

{

    std::vector<std::string> lexems;

    std::string tmp = "";

    switch (type)

    {

    case INFIX:

        for (char c : inp) {

            if (std::regex\_match(std::string(1, c), std::regex("[\\w\\d\\.\\,]"))) {

                if (c == ',')

                    tmp += '.';

                else

                    tmp += c;

            }

            else {

                if (tmp != "")

                    lexems.push\_back(tmp);

                if (c != 32) {

                    if ((c == '-') && (lexems.size() == 0 || priority.count(lexems.back()))) {

                        lexems.push\_back("~");

                    }

                    else if (priority.count(std::string(1, c))) {

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

                    }

                    else {

                        throw std::logic\_error(

                            std::format("Unresolve symbol at {}/{}", this->infix.find(std::string(1, c)) + 1, this->infix.size()));

                    }

                }

                tmp = "";

            }

        }

        if (tmp != "")

            lexems.push\_back(tmp);

        break;

    case POSTFIX:

        std::stringstream sinp(inp);

        while (sinp >> tmp)

            lexems.push\_back(tmp);

        break;

    }

    return lexems;

}

void TArithmeticExpression::ToPostfix() {

    auto lexems = Parse(infix, INFIX);

    TStack<std::string> st(lexems.size());

    std::string stackItem;

    for (std::string item : lexems) {

        if (std::regex\_match(item, std::regex("[\\(\\[\\{]"))) {

            st.Push(item);

        }

        else if (std::regex\_match(item, std::regex("[\\)\\]\\}]"))) {

            stackItem = st.TopView();

            st.Pop();

            std::string comp;

            if (item == ")") comp = "(";

            else if (item == "]") comp = "[";

            else comp = "{";

            while (stackItem != comp) {

                postfix += stackItem + " ";

                stackItem = st.TopView();

                st.Pop();

            }

        }

        else if (std::regex\_match(item, std::regex("[\\+\\-\\\*\\/\\!\\~\\^]|(sin)|(cos)|(tg)|(ctg)"))) {

            while (!st.IsEmpty()) {

                stackItem = st.TopView();

                st.Pop();

                if (priority.at(item) <= priority.at(stackItem))

                    postfix += stackItem + " ";

                else {

                    st.Push(stackItem);

                    break;

                }

            }

            st.Push(item);

        }

        else {

            if (item == "PI") {

                postfix += item + " ";

            }

            else {

                try {

                    double \_ = std::stod(item);

                }

                catch (std::invalid\_argument e) {

                    operands.insert({ item, 0.0 });

                }

                postfix += item + " ";

            }

        }

    }

    while (!st.IsEmpty()) {

        stackItem = st.TopView();

        st.Pop();

        postfix += stackItem + " ";

    }

}

void TArithmeticExpression::ToInfix()

{

    auto lexems = Parse(infix, INFIX);

    infix = "";

    for (auto lexem : lexems) {

        if (lexem == "~")

            infix += lexem;

        else

            infix += lexem + " ";

    }

}

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

{

    std::vector<std::string> op;

    for (const auto& item : operands)

        op.push\_back(item.first);

    return op;

}

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

{

    for (auto& val : values)

    {

        try

        {

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

        }

        catch (std::out\_of\_range&) {}

    }

    double leftOperand, rightOperand;

    auto postfixLexems = Parse(postfix, POSTFIX);

    TStack<double> st(postfixLexems.size() + values.size());

    for (std::string lexem : postfixLexems )

    {

        if (std::regex\_match(lexem, std::regex("[\\+\\-\\/\\\*\\^]"))) {

            double res = 0;

            rightOperand = st.TopView();

            st.Pop();

            leftOperand = st.TopView();

            st.Pop();

            if (lexem == "+") res = leftOperand + rightOperand;

            if (lexem == "-") res = leftOperand - rightOperand;

            if (lexem == "\*") res = leftOperand \* rightOperand;

            if (lexem == "^") res = std::pow(leftOperand, rightOperand);

            if (lexem == "/") {

                if (rightOperand == 0)

                    throw std::logic\_error("Divition by zero");

                res = leftOperand / rightOperand;

            }

            st.Push(res);

        }

        else if (std::regex\_match(lexem, std::regex("[\\!\\~]|(sin)|(cos)|(tg)|(ctg)")))

        {

            leftOperand = st.TopView();

            st.Pop();

            if (lexem == "!")

                st.Push(std::tgamma(leftOperand + 1));

            if (lexem == "~") st.Push(-1 \* leftOperand);

            if (lexem == "sin") st.Push(std::sin(leftOperand));

            if (lexem == "cos") st.Push(std::cos(leftOperand));

            if (lexem == "tg") st.Push(std::tan(leftOperand));

            if (lexem == "ctg") st.Push(std::cos(leftOperand) / std::sin(leftOperand));

        }

        else if (lexem == "PI") {

            st.Push(PI);

        }

        else {

            try {

                st.Push(std::stod(lexem));

            }

            catch (std::invalid\_argument e){

                st.Push(operands[lexem]);

            }

        }

    }

    return st.TopView();

}

void TArithmeticExpression::IsValidExpression() {

    TStack<std::string> stack(this->infix.size());

    auto lexems = Parse(this->infix, INFIX);

    int operandBalance = 0;

    for (std::string lexem : lexems) {

        if (std::regex\_match(lexem, std::regex("[\\(\\{\\[]"))) {

            stack.Push(lexem);

        }

        else if (std::regex\_match(lexem, std::regex("[\\)\\]\\}]"))) {

            std::string openBracket;

            if (lexem == ")") openBracket = "(";

            if (lexem == "]") openBracket = "[";

            if (lexem == "}") openBracket = "{";

            if (stack.IsEmpty() || stack.TopView() != openBracket) {

                // Несоответствие скобок

                throw std::logic\_error(

                    std::format("Wrong bracket {} at {}/{}", lexem, this->infix.find(lexem)+1, this->infix.size()));

            }

            stack.Pop();

        }

        else if (priority.count(lexem)) {

            // Операторы допустимы

            if (std::regex\_match(lexem, std::regex("[\\+\\-\\/\\\*\\^]")))

                operandBalance--;

            continue;

        }

        else if (std::regex\_match(lexem, std::regex("[\\d\\.\\w]+"))) {

            // Допустимые символы: буквы, цифры и точка (для десятичных чисел)

            operandBalance++;

            continue;

        }

        else {

            // Недопустимый символ

            throw std::logic\_error(

                std::format("Unresolved symbol {} at {}/{}", lexem, this->infix.find(lexem) + 1, this->infix.size()));

        }

    }

    // Проверка на оставшиеся открывающие скобки

    if (!stack.IsEmpty()) {

        std::string closeBracket, lexem = stack.TopView();

        if (lexem == "(") closeBracket = ")";

        if (lexem == "[") closeBracket = "]";

        if (lexem == "{") closeBracket = "}";

        throw std::logic\_error(

            std::format("Expected {0} at {1}/{1}", closeBracket, this->infix.size()) + "\n");

    }

    // Проверка на количество операндов

    if (operandBalance < 1) {

        throw std::logic\_error("Too few operands");

    }

    else if (operandBalance > 1) {

        throw std::logic\_error("Too many operands");

    }

}

1. Приложение 7

Priority.cpp

#include "MyExpression.h"

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

    {"(", 0},

    {"[", 0},

    {"{", 0},

    {")", 0},

    {"]", 0},

    {"}", 0},

    {"+", 1},

    {"-", 1},

    {"\*", 2},

    {"/", 2},

    {"^", 2},

    {"~", 3},

    {"!", 3},

    {"sin", 4},

    {"cos", 4},

    {"tg", 4},

    {"ctg", 4},

};

1. Приложение 8

test\_expression.h

#include "MyExpression.h"

#include <gtest.h>

using namespace std;

TEST(TArithmeticExpression, not\_a\_word\_in\_Russian)

{

    string expr = "�";

    ASSERT\_ANY\_THROW(TArithmeticExpression expression(expr));

}

TEST(TArithmeticExpression, can\_get\_operands)

{

    string expr = "a+b";

    TArithmeticExpression expression(expr);

    vector<string> v;

    v.push\_back("a");

    v.push\_back("b");

    ASSERT\_EQ(v, expression.GetOperands());

}

TEST(TArithmeticExpression, can\_get\_infix)

{

    string expr = "a+b";

    TArithmeticExpression expression(expr);

    string str = "a + b ";

    ASSERT\_EQ(str, expression.GetInfix());

}

TEST(TArithmeticExpression, can\_get\_postfix)

{

    string expr = "a+b";

    TArithmeticExpression expression(expr);

    string str = "a b + ";

    ASSERT\_EQ(str, expression.GetPostfix());

}

TEST(TArithmeticExpression, can\_calculate)

{

    string expr = "1+1";

    TArithmeticExpression expression(expr);

    map<string, double> values;

    ASSERT\_EQ(2, expression.Calculate(values));

}

TEST(TArithmeticExpression, lesson\_example)

{

    string expr = "(4+11-8/2\*(7\*3+4-7))\*3";

    TArithmeticExpression expression(expr);

    map<string, double> values;

    EXPECT\_EQ(-171.0, expression.Calculate(values));

}

TEST(TArithmeticExpression, priority)

{

    string expr = "-sin(PI/2)+2+2\*2/2!^2";

    TArithmeticExpression expression(expr);

    map<string, double> values;

    EXPECT\_EQ(5, round(expression.Calculate(values) \* 100) / 100);

}

TEST(TArithmeticExpression, calculate\_sin)

{

    string expr = "sin(PI/2)";

    TArithmeticExpression expression(expr);

    map<string, double> values;

    EXPECT\_EQ(1, round(expression.Calculate(values) \* 100) / 100);

}

TEST(TArithmeticExpression, calculate\_cos)

{

    string expr = "cos(PI/2)";

    TArithmeticExpression expression(expr);

    map<string, double> values;

    EXPECT\_EQ(0, round(expression.Calculate(values) \* 100) / 100);

}

TEST(TArithmeticExpression, calculate\_tg)

{

    string expr = "tg(PI/4)";

    TArithmeticExpression expression(expr);

    map<string, double> values;

    EXPECT\_EQ(1, round(expression.Calculate(values) \* 100) / 100);

}

TEST(TArithmeticExpression, calculate\_ctg)

{

    string expr = "ctg(PI/4)";

    TArithmeticExpression expression(expr);

    map<string, double> values;

    EXPECT\_EQ(1, round(expression.Calculate(values) \* 100) / 100);

}

TEST(TArithmeticExpression, calculate\_factorial)

{

    string expr = "5!";

    TArithmeticExpression expression(expr);

    map<string, double> values;

    EXPECT\_EQ(120, expression.Calculate(values));

}

TEST(TArithmeticExpression, unary\_minus)

{

    string expr = "-1";

    TArithmeticExpression expression(expr);

    ASSERT\_EQ("-1 ", expression.GetInfix());

    ASSERT\_EQ("1 ~ ", expression.GetPostfix());

    map<string, double> values;

    EXPECT\_EQ(-1, expression.Calculate(values));

}

TEST(TArithmeticExpression, multicharacter\_lexem)

{

    string expr = "ans+row";

    TArithmeticExpression expression(expr);

    ASSERT\_EQ("ans + row ", expression.GetInfix());

    ASSERT\_EQ("ans row + ", expression.GetPostfix());

}

TEST(TArithmeticExpression, another\_brackets)

{

    string expr = "([{ans+row}])";

    ASSERT\_NO\_THROW(TArithmeticExpression expression(expr));

    TArithmeticExpression expression(expr);

    ASSERT\_EQ("( [ { ans + row } ] ) ", expression.GetInfix());

    ASSERT\_EQ("ans row + ", expression.GetPostfix());

}

TEST(TArithmeticExpression, isValidExpression)

{

    string expr = "([{ans+row]})";

    ASSERT\_ANY\_THROW(TArithmeticExpression expression(expr));

    expr = "([{ans+row})";

    ASSERT\_ANY\_THROW(TArithmeticExpression expression(expr));

    expr = "ans+row&)";

    ASSERT\_ANY\_THROW(TArithmeticExpression expression(expr));

    expr = "ans+row row";

    ASSERT\_ANY\_THROW(TArithmeticExpression expression(expr));

    expr = "ans+";

    ASSERT\_ANY\_THROW(TArithmeticExpression expression(expr));

}

1. Приложение 9

Expression\_sample.h

#include <iostream>

#include "MyExpression.h"

using namespace std;

int main()

{

    try {

        string input;

        cout << "Enter expression: ";

        cin >> input;

        TArithmeticExpression expression(input);

        cout << expression.GetInfix() << endl;

        cout << expression.GetPostfix() << endl;

        vector<string> operands = expression.GetOperands();

        map<string, double> values;

        double value;

        for (const auto& op : operands)

        {

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

            cin >> value;

            values[op] = value;

        }

        cout << expression.Calculate(values) << endl;

    }

    catch (exception e) {

        cerr << e.what();

    }

}

1. Приложение 10

Expression\_experiments.cpp

#include <iostream>

#include <chrono>

#include <random>

#include <map>

#include "MyExpression.h"

#include "../../stack/include/MyStack.h"

auto average\_test(size\_t size, size\_t iterations = 10) {

    int max\_random = 10000;

    int min\_random = -10000;

    long long average\_time\_add = 0;

    long long average\_time\_del = 0;

    for (size\_t i = 0; i < iterations; i++) {

        TStack<int> stack(size);

        std::cout << "#" << i << " Adding" << std::endl;

        for (size\_t j = 0; j < size; j++) {

            int c1 = min\_random + std::rand() % static\_cast<int>(max\_random - min\_random + 1);

            std::chrono::steady\_clock::time\_point begin, end;

            begin = std::chrono::steady\_clock::now();

            stack.Push(c1);

            end = std::chrono::steady\_clock::now();

            auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

            average\_time\_add += elapsed\_ms.count();

            std::cout << i + 1 << '\t' << elapsed\_ms.count() << std::endl;

        }

        std::cout << std::endl << "#" << i << " Delliting" << std::endl;

        for (size\_t j = 0; j < size; j++) {

            std::chrono::steady\_clock::time\_point begin, end;

            begin = std::chrono::steady\_clock::now();

            stack.Pop();

            end = std::chrono::steady\_clock::now();

            auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

            average\_time\_add += elapsed\_ms.count();

            std::cout << i + 1 << '\t' << elapsed\_ms.count() << std::endl;

        }

    }

    std::cout << "Average time add: " << average\_time\_add / static\_cast<double>(iterations \* size) << std::endl;

    std::cout << "Average time del: " << average\_time\_add / static\_cast<double>(iterations \* size) << std::endl;

    return;

}

int main(int argc, char\*\* arhv)

{

    average\_test(150);

    auto begin = std::chrono::steady\_clock::now();

    std::string expr = "(4+11-8/2\*(7\*3+4-7))\*3";

    TArithmeticExpression expression(expr);

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

    expression.Calculate(values);

    auto end = std::chrono::steady\_clock::now();

    auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

    std::cout << "Execution time: " << elapsed\_ms.count();

}

1. Приложение 11

MyForm.cpp

#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void Main(array<String^>^ args)

{

    Application::EnableVisualStyles();

    Application::SetCompatibleTextRenderingDefault(false);

    CppWinForm1::MyForm form;

    Application::Run(% form);

}

1. Приложение 12

MyForm.h

#pragma once

#include "MyExpression.h"

#include "MyForm1.h"

#include <msclr\marshal\_cppstd.h>

namespace CppWinForm1 {

    using namespace System;

    using namespace System::ComponentModel;

    using namespace System::Collections;

    using namespace System::Windows::Forms;

    using namespace System::Data;

    using namespace System::Drawing;

    /// <summary>

    /// Summary for MyForm

    /// </summary>

    public ref class MyForm : public System::Windows::Forms::Form

    {

    public:

        MyForm(void)

        {

            InitializeComponent();

        }

    private: System::Windows::Forms::TextBox^ textBox1;

    private: System::Windows::Forms::Button^ button1;

    private: System::Windows::Forms::Label^ label1;

    protected:

        /// <summary>

        /// Clean up any resources being used.

        /// </summary>

        ~MyForm()

        {

            if (components)

            {

                delete components;

            }

        }

    private:

        /// <summary>

        /// Required designer variable.

        /// </summary>

        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

        /// <summary>

        /// Required method for Designer support - do not modify

        /// the contents of this method with the code editor.

        /// </summary>

        void InitializeComponent(void)

        {

            this->textBox1 = (gcnew System::Windows::Forms::TextBox());

            this->button1 = (gcnew System::Windows::Forms::Button());

            this->label1 = (gcnew System::Windows::Forms::Label());

            this->SuspendLayout();

            //

            // textBox1

            //

            this->textBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 20, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

                static\_cast<System::Byte>(204)));

            this->textBox1->Location = System::Drawing::Point(12, 12);

            this->textBox1->Name = L"textBox1";

            this->textBox1->Size = System::Drawing::Size(466, 53);

            this->textBox1->TabIndex = 1;

            this->textBox1->KeyPress += gcnew System::Windows::Forms::KeyPressEventHandler(this, &MyForm::CheckEnterKeyPress);

            //

            // button1

            //

            this->button1->Location = System::Drawing::Point(484, 11);

            this->button1->Name = L"button1";

            this->button1->Size = System::Drawing::Size(204, 60);

            this->button1->TabIndex = 2;

            this->button1->Text = L"Enter Expression";

            this->button1->UseVisualStyleBackColor = true;

            this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

            //

            // label1

            //

            this->label1->AutoSize = true;

            this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 20, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

                static\_cast<System::Byte>(204)));

            this->label1->Location = System::Drawing::Point(12, 71);

            this->label1->Name = L"label1";

            this->label1->Size = System::Drawing::Size(0, 46);

            this->label1->TabIndex = 3;

            //

            // MyForm

            //

            this->AutoScaleDimensions = System::Drawing::SizeF(9, 20);

            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

            this->ClientSize = System::Drawing::Size(700, 250);

            this->Controls->Add(this->label1);

            this->Controls->Add(this->button1);

            this->Controls->Add(this->textBox1);

            this->Name = L"MyForm";

            this->Text = L"Expression Calculator";

            this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

            this->ResumeLayout(false);

            this->PerformLayout();

        }

#pragma endregion

    private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

    }

    private: void CheckEnterKeyPress(System::Object^ sender, System::Windows::Forms::KeyPressEventArgs^ e)

    {

        if (e->KeyChar == (char)Keys::Return)

        {

            button1\_Click(sender, e);

        }

    }

    private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

        msclr::interop::marshal\_context context;

        std::string expr = context.marshal\_as<std::string>(textBox1->Text);

        std::string text = "";

        try {

            TArithmeticExpression expression(expr);

            text += "Infix: " + expression.GetInfix() + '\n';

            text += "Postfix: " + expression.GetPostfix() + '\n';

            auto tmp = expression.GetOperands();

            std::map<std::string, double> tmp1;

            double value = 0.0;

            for (auto t : tmp) {

                CppWinForm1::MyForm1^ form1 = gcnew CppWinForm1::MyForm1(t, value);

                form1->ShowDialog();

                tmp1[t] = value;

            }

            text += std::format("Calculated: {:.2f}", expression.Calculate(tmp1));

            label1->Text = gcnew String(text.c\_str());

        }

        catch (std::exception e) {

            MessageBox::Show(gcnew String(e.what()));

        }

    }

};

}

1. Приложение 13

MyForm1.h

#pragma once

#include <msclr\marshal\_cppstd.h>

namespace CppWinForm1 {

    using namespace System;

    using namespace System::ComponentModel;

    using namespace System::Collections;

    using namespace System::Windows::Forms;

    using namespace System::Data;

    using namespace System::Drawing;

    /// <summary>

    /// Summary for MyForm1

    /// </summary>

    public ref class MyForm1 : public System::Windows::Forms::Form

    {

    public:

        MyForm1(const std::string& lexem, double& out) :out(out)

        {

            InitializeComponent();

            this->lexem = gcnew String(lexem.c\_str());

        }

    private: System::Windows::Forms::TextBox^ textBox1;

    private: System::Windows::Forms::Button^ button1;

    private: System::Windows::Forms::Label^ label2;

    private: double& out;

    private: System::String^ lexem;

    protected:

        /// <summary>

        /// Clean up any resources being used.

        /// </summary>

        ~MyForm1()

        {

            if (components)

            {

                delete components;

            }

        }

    private:

        /// <summary>

        /// Required designer variable.

        /// </summary>

        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

        /// <summary>

        /// Required method for Designer support - do not modify

        /// the contents of this method with the code editor.

        /// </summary>

        void InitializeComponent(void)

        {

            this->textBox1 = (gcnew System::Windows::Forms::TextBox());

            this->button1 = (gcnew System::Windows::Forms::Button());

            this->label2 = (gcnew System::Windows::Forms::Label());

            this->SuspendLayout();

            //

            // textBox1

            //

            this->textBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 20, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

                static\_cast<System::Byte>(204)));

            this->textBox1->Location = System::Drawing::Point(20, 104);

            this->textBox1->Name = L"textBox1";

            this->textBox1->Size = System::Drawing::Size(466, 53);

            this->textBox1->TabIndex = 1;

            this->textBox1->KeyPress += gcnew System::Windows::Forms::KeyPressEventHandler(this, &MyForm1::CheckEnterKeyPress);

            //

            // button1

            //

            this->button1->Location = System::Drawing::Point(492, 97);

            this->button1->Name = L"button1";

            this->button1->Size = System::Drawing::Size(204, 60);

            this->button1->TabIndex = 2;

            this->button1->Text = L"Enter Expression";

            this->button1->UseVisualStyleBackColor = true;

            this->button1->Click += gcnew System::EventHandler(this, &MyForm1::button1\_Click);

            //

            // label2

            //

            this->label2->AutoSize = true;

            this->label2->Location = System::Drawing::Point(26, 17);

            this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 20, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

                static\_cast<System::Byte>(204)));

            this->label2->Name = L"label2";

            this->label2->Size = System::Drawing::Size(51, 20);

            this->label2->TabIndex = 4;

            this->label2->Text = L"label2";

            //

            // MyForm1

            //

            this->AutoScaleDimensions = System::Drawing::SizeF(9, 20);

            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

            this->ClientSize = System::Drawing::Size(700, 250);

            this->Controls->Add(this->label2);

            this->Controls->Add(this->button1);

            this->Controls->Add(this->textBox1);

            this->Name = L"MyForm1";

            this->Text = L"Value Manager";

            this->Load += gcnew System::EventHandler(this, &MyForm1::MyForm1\_Load);

            this->ResumeLayout(false);

            this->PerformLayout();

        }

#pragma endregion

    private: System::Void MyForm1\_Load(System::Object^ sender, System::EventArgs^ e) {

        label2->Text = "Enter value for lexem \"" + lexem + "\": ";

    }

    private: void CheckEnterKeyPress(System::Object^ sender, System::Windows::Forms::KeyPressEventArgs^ e)

    {

        if (e->KeyChar == (char)Keys::Return)

        {

            button1\_Click(sender, e);

        }

    }

    private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

        msclr::interop::marshal\_context context;

        std::string value = context.marshal\_as<std::string>(textBox1->Text);

        try {

            out = std::stod(value);

        }

        catch (std::exception e) {

            out = 0.0;

        }

        this->Hide();

    }

};

}

1. Приложение 14

test\_main.cpp

#include <gtest.h>

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}