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: «Фундаментальная информатика и информационные технологии»

**ОТЧЕТ**

по лабораторной работе

**ВЫЧИСЛЕНИЕ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ**

**Выполнил:** студент группы

382006-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Н. Грачёв

Подпись

**Проверил:** к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.А. Баркалов

Подпись

Нижний Новгород  
2021

Содержание:

[1. Введение 2](#_Toc89184669)

[2. Постановка задачи 2](#_Toc89184670)

[3. Описание алгоритмов 2](#_Toc89184671)

[Алгоритм работы динамической структуры хранения данных «Стек» 3](#_Toc89184672)

[При создании стека 3](#_Toc89184673)

[Добавление элемента в стек 4](#_Toc89184674)

[Извлечение элемента из стека 4](#_Toc89184675)

[Посмотреть элемент на вершине стека 4](#_Toc89184676)

[Узнать, переполнен ли стек 4](#_Toc89184677)

[Узнать, пуст ли стек 4](#_Toc89184678)

[Очистка стека 4](#_Toc89184679)

[Алгоритм проверки корректности введенного арифметического выражения 5](#_Toc89184680)

[Проверка на расстановку скобок: 5](#_Toc89184681)

[Проверка на расстановку операций: 5](#_Toc89184682)

[Проверка на наличие запрещенных символов: 5](#_Toc89184683)

[Алгоритм составления постфиксной записи выражения 6](#_Toc89184684)

[Нахождение значения выражения с использованием постфиксной записи 7](#_Toc89184685)

[Нахождение значения выражения без использования постфиксной записи 7](#_Toc89184686)

[4. Описание программы 8](#_Toc89184687)

[Класс «TCalculator» 9](#_Toc89184688)

[Класс «TStack» 10](#_Toc89184689)

[Тестирование работы классов при помощи Google Tests 11](#_Toc89184690)

[Тестирование класса «TStack» 11](#_Toc89184691)

[Тестирование класса «TCalculator» 11](#_Toc89184692)

[Графический интерфейс 12](#_Toc89184693)

[5. Результаты 14](#_Toc89184694)

[6. Заключение 14](#_Toc89184695)

[7. Литература 14](#_Toc89184696)

[8. Приложение 15](#_Toc89184697)

[TCalculator.h 15](#_Toc89184698)

[TStack.h 20](#_Toc89184699)

[GTest\_TStack.cpp 22](#_Toc89184700)

[GTest\_Calculator.cpp 24](#_Toc89184701)

[CForm.h 26](#_Toc89184702)

# 

# Введение

Во время решения различных задач нам всегда необходимо вычислять промежуточные и итоговые результаты. В Интернете имеется большое количество сервисов, которые умеют вычислять сложные математические выражения, например, Wolfram Alpha. В данном случае речь будет идти о написании программы-калькулятора, которая бы смогла находить значение арифметических выражений.

# Постановка задачи

Задача: Разработать программу-калькулятор, которая вычисляет значение арифметического выражения, заданного в виде строки. Калькулятор должен работать с операциями сложения, вычитания, умножения, деления, возведения в степень. Для данной программы необходимо написать Google Test`ы и проверить корректность ее работы. Окончательный вариант программы оформить при помощи графического интерфейса Windows Forms: в первое текстовое поле пользователь вводит арифметическое выражение и, при нажатии на кнопку, во втором текстовом поле выводится результат вычислений.

# Описание алгоритмов

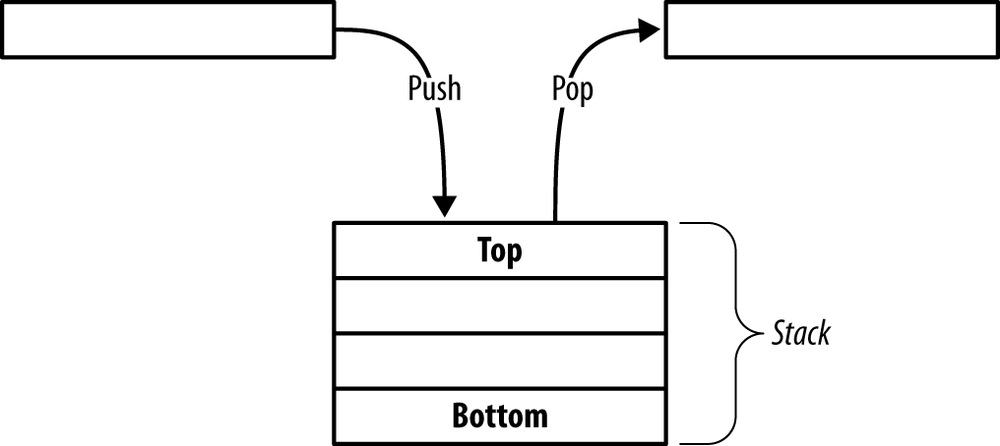
В программе действуют четыре основных алгоритма:

* Проверка корректности введенного арифметического выражения;
* Составление постфиксной записи выражения;
* Нахождение значения выражения с использованием постфиксной записи;
* Нахождение значения выражения без использования постфиксной записи, «на лету»;

Данные алгоритмы используют динамическую структуру хранения данных «Стек», поэтому еще одним дополнительным алгоритмом является алгоритм работы стека.

## Алгоритм работы динамической структуры хранения данных «Стек»

Стек устроен по принципу LIFO (Last in first out) – мы можем извлекать из стека только последний добавленный в него элемент. Для того чтобы извлечь или посмотреть элемент, находящийся не на вершине стека, мы должны извлечь все элементы, которые были добавлены после этого элемента.



Необходимо реализовать следующие функции:

* Push – Добавить элемент в стек,
* Pop – Извлечь последний элемент,
* Top – Посмотреть последний элемент,
* Full – Проверка, заполнен ли стек,
* Empty – Проверка на пустоту стека,
* Clear – Очистить стек.

### При создании стека

При создании стека необходимо указать его размер, выделить память под соответствующее количество элементов. По умолчанию стек пуст, и переменной, назовем ее top, хранящей позицию последнего элемента в массиве, присвоим значение -1.

### Добавление элемента в стек

Чтобы добавить элемент в стек, мы должны увеличить значение переменной top на единицу и затем элементу массива с индексом top присвоить нужное значение.

### Извлечение элемента из стека

Чтобы извлечь элемент из стека, мы должны вернуть элемент, находящийся на вершине, а затем уменьшить значение с позицией последнего элемента на единицу.

### Посмотреть элемент на вершине стека

Просто возвращаем элемент массива, находящийся в массиве на позиции top.

### Узнать, переполнен ли стек

Если top равен размеру массива минус один, то стек переполнен. Иначе – еще есть место.

### Узнать, пуст ли стек

Если top равен -1, то стек пуст.

### Очистка стека

Присваиваем переменной top значение -1.

## Алгоритм проверки корректности введенного арифметического выражения

Выражение может быть задано неверно. В основном, неверные выражения имеют 3 вида:

* Скобки расставлены неверно, например: )3+5(
* Операции расставлены неверно: 3+-4
* В выражении имеются запрещенные символы: 3a +4,.B3@

### Проверка на расстановку скобок:

Создадим стек и будем проходить циклом наше выражение слева-направо. Если встречаем открывающую скобку – кладем ее в стек. Если встретили закрывающую – удаляем последний элемент из стека. Если в процессе такой проверки не было выброшено исключений и стек в итоге остался пустым, значит, выражение задано верно.

### Проверка на расстановку операций:

Если проигнорировать скобки, то можно заметить, что любое арифметическое выражение - это последовательность, которая начинается с числа, а затем операция и число чередуются, заканчивается выражение числом. Циклом пробегаем по всему выражению и проверяем, является ли заданное выражение таковой последовательностью. Если нет – значит, выражение задано неверно.

### Проверка на наличие запрещенных символов:

Проверяем каждый символ выражения, является ли он одним из разрешенных символов. К разрешенным относятся – числа (в т. ч. десятичные и в экспоненциальной форме), скобки, операции +, -, \, \*, знаки пробела. Если какой-либо символ не относится к данной группе, то выражение задано неверно.

Стоит отметить, что символы «е» и «.», которые используются для записи десятичных чисел и их экспоненциальной формы также являются запрещенными. Однако при проверке на то, является ли какая-либо часть строки числом, можно воспользоваться функцией C++ stod(), которая считает число в любой форме, после считывания числа переместит текущий элемент проверки уже за символы «е» и «.» так, что они не попадут под проверку на запрещенные символы.

## Алгоритм составления постфиксной записи выражения

Чтобы перевести стандартную запись выражения в постфиксную, необходимо:

1. Ввести для операций приоритет:  
   Для + и – приоритет равен 1, для \* и \ приоритет равен 2, для возведения в степень приоритет равен 3. Для открывающих скобок приоритет равен 0.
2. Создать стек для хранения операций.
3. Заключить исходное выражение в скобки и начать просматривать его слева-направо.
4. Числа, по мере их появления, добавляются в строку-результат.
5. Левые скобки всегда сразу помещаются в стек.
6. С операциями поступаем следующим образом:  
   Если приоритет данной операции выше той, что лежит на вершине стека – тогда добавляем ее в стек.  
   Если приоритет ниже или равен приоритету операции, лежащей на вершине стека, то достаем из стека операции и приписываем их к строке-результату до тех пор, пока приоритет исходной операции не станет выше.
7. Закрывающая скобка заставляет приписать к строке-результату все извлеченные операции, лежащие в стеке после закрывающей скобки, а оставшуюся открывающую скобку удалить из стека.

На выходе получим постфиксную запись, которая применяется для вычисления выражения за один проход.

## Нахождение значения выражения с использованием постфиксной записи

Используя ранее заготовленную постфиксную запись выражения, его значение можно вычислить следующим путем:

1. Создаем стек для чисел.
2. Просматриваем постфиксную запись слева-направо.
3. Если встречаем число – добавляем его в стек.
4. Если встречаем операцию – достаем два числа из стека: первое извлеченное станет вторым операндом, а второе извлеченное станет первым операндом. Выполняем операцию и кладем результат в стек.

После того, как пройдем по всей постфиксной записи, в стеке останется одно число. Это число и есть результат выражения.

## Нахождение значения выражения без использования постфиксной записи

Для данного алгоритма понадобятся два стека – стек для чисел и стек для операций. Перед вычислением необходимо ввести приоритет операций (для открывающей скобки приоритет всегда равен 0), а затем заключить исходное выражение в скобки. Просматривая строку посимвольно, действуем следующим образом:

1. Если встречается число – кладем его в стек с числами (с помощью функции stod() мы сможем считать числа, состоящие из нескольких символов),
2. Открывающие скобки кладем в стек без сравнения приоритетов,
3. Если встретилась более приоритетная операция, чем та, что лежит на вершине стека с операциями – добавляем ее в этот стек,
4. Если встретилась менее или равно приоритетная операция – запоминаем ее, извлекаем из стека и выполняем операции над двумя извлеченными операндами из стека с числами, кладя результат операции в стек с числами. Так делаем до тех пор, пока приоритет операции, которую мы запомнили ранее, не станет выше той, что оказалась на вершине стека с операциями. Кладем эту операцию в стек.
5. Если встретили закрывающую скобку – извлекаем и выполняем над операндами из стека с числами операции до тех пор, пока на вершине стека с операциями не окажется закрывающая скобка. Удаляем закрывающую скобку из стека.

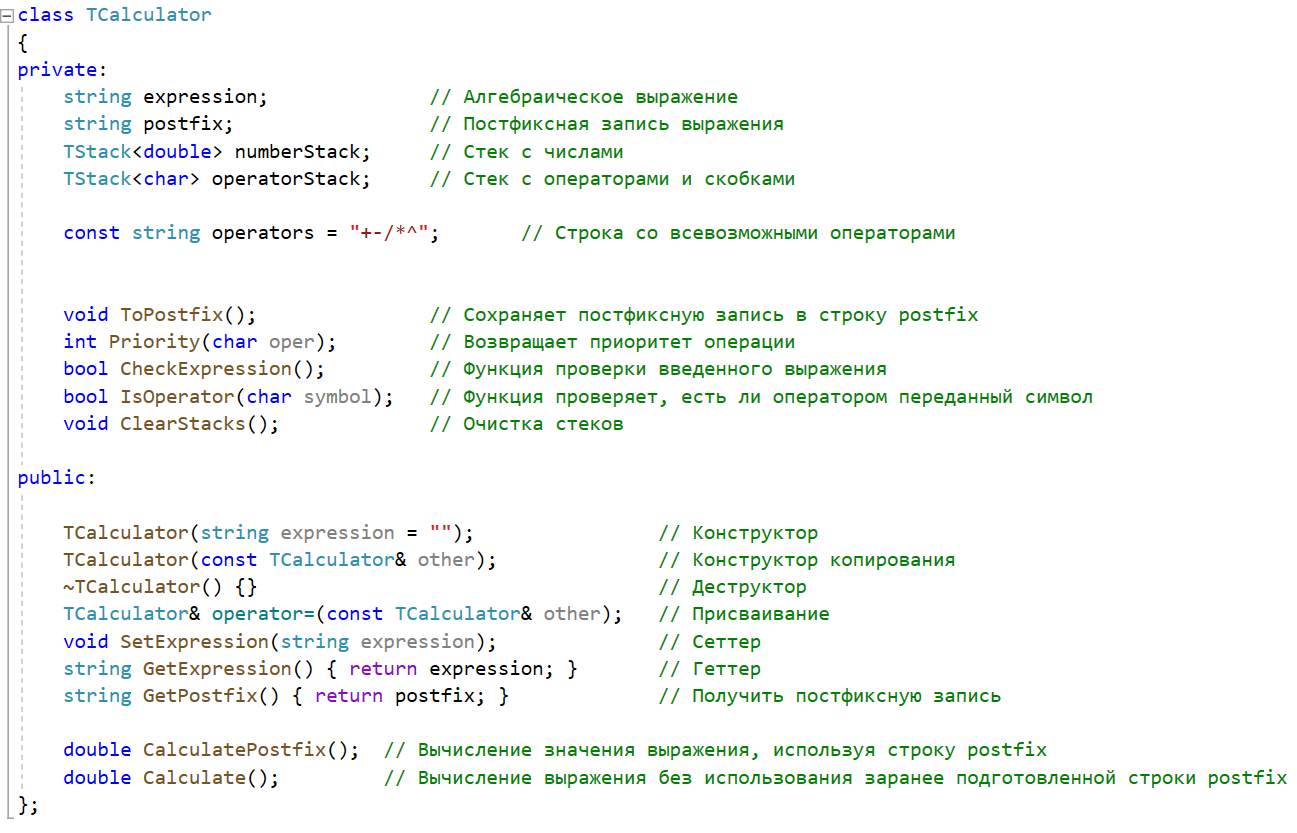
В результате такого алгоритма, после прохода по всему выражению в стеке с числами должно остаться одно число – значение выражения.

# Описание программы

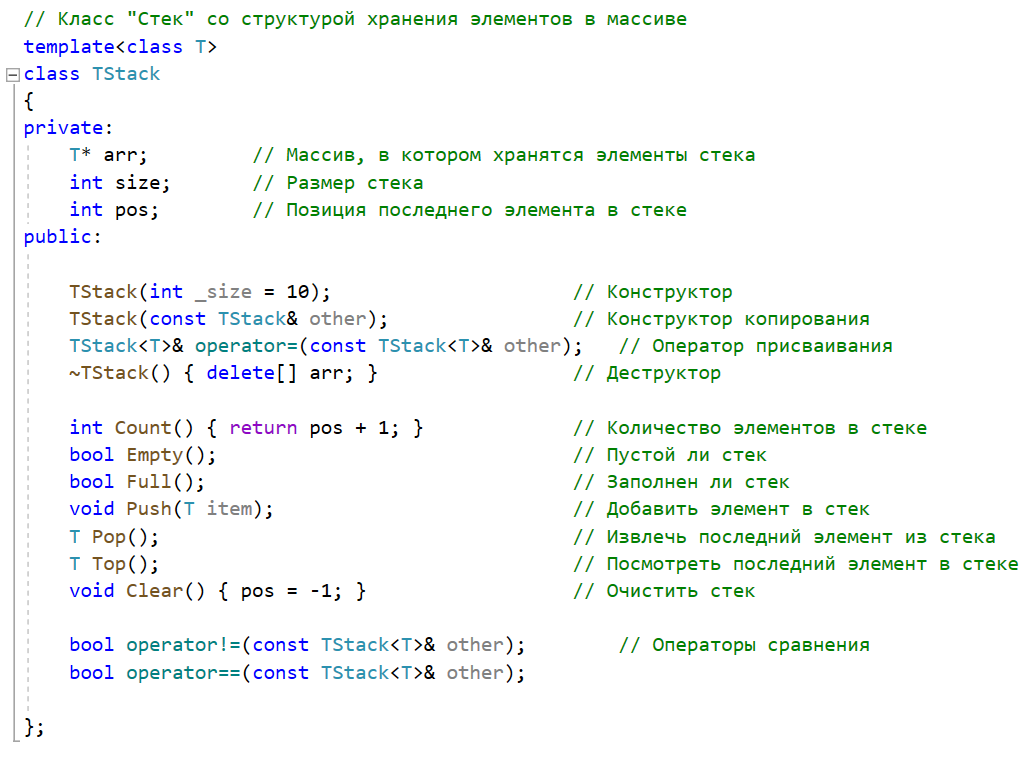
Программа состоит из трех частей:

* Файлы с классами «TCalculator» и «TStack»,
* Тестирование работы классов при помощи Google Tests,
* Графический интерфейс программы, реализованный на Windows Forms.

## Класс «TCalculator»

Класс TCalculator находится в файле «TCalculator.h» и в нем имеются следующие поля и методы:

## Класс «TStack»

Шаблонный класс TStack находится в файле «TStack.h» и в нем имеются следующие поля и методы:

## Тестирование работы классов при помощи Google Tests

### Тестирование класса «TStack»

Для проверки корректности работы класса были написаны следующие тесты:

* Можем создать стек с положительным числом элементов,
* Не можем создать стек с отрицательным числом элементов,
* Стек после создания является пустым,
* Проверка работы оператора ==,
* Проверка, имеет ли стек собственную память,
* Можем положить элемент в стек,
* Стек не пустой после добавления элемента,
* Стек становится пустым после извлечения единственного элемента,
* Стек полон, когда не остается места под новые элементы,
* Не можем положить элемент в переполненный стек,
* Не можем извлечь элемент из пустого стека,
* Проверка работы счетчика элементов,
* Стек становится пустым после работы метода Clear().

### Тестирование класса «TCalculator»

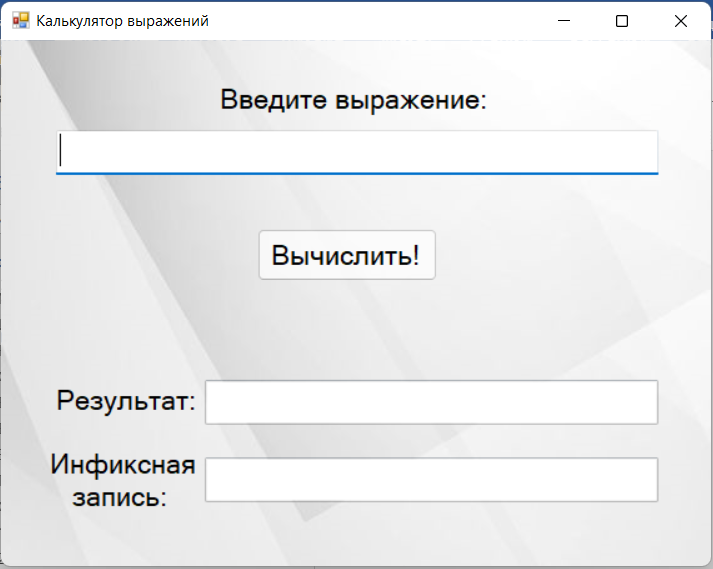
Для данного класса были написаны следующие тесты:

* Можем создать калькулятор с корректным выражением,
* Не можем создать калькулятор с некорректным выражением (тестируются всевозможные виды некорректных выражений),
* 3 теста с разными выражениями, имеющие только целые числа,
* 3 теста с разными выражениями, имеющие десятичные дроби,
* 3 теста с разными выражениями, имеющие числа в экспоненциальной форме.

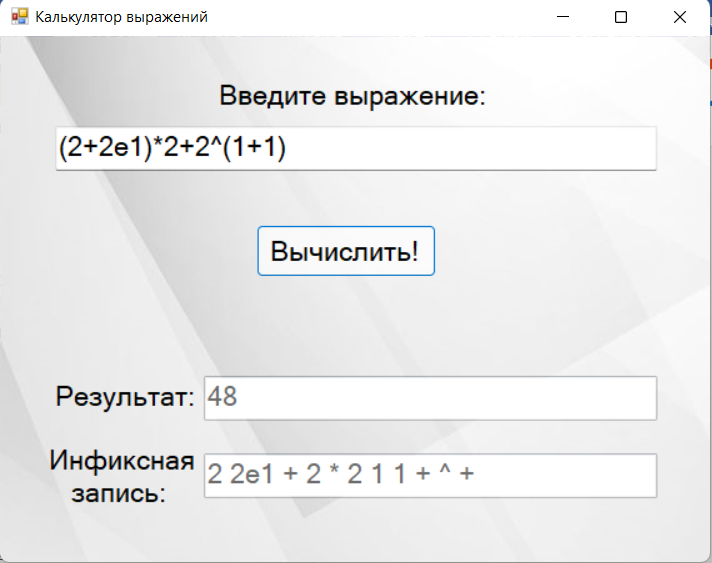
Все тесты успешно пройдены, классы отлажены и работают корректно.

## Графический интерфейс

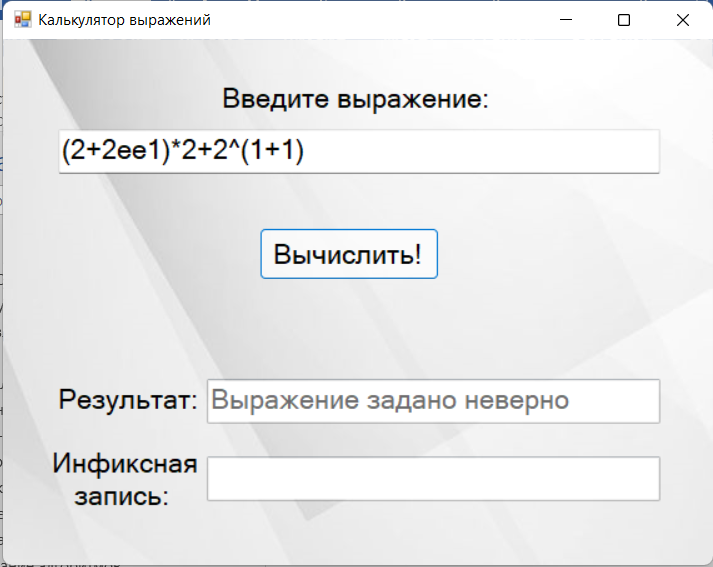
Графический интерфейс, реализованный с помощью Windows Forms.



В первую строку пользователь вводит выражение. Далее необходима нажать кнопку «Вычислить!». Если пользователь задал корректное выражение, то в строке «Результат» и «Инфиксная запись» появятся соответствующие значения:



Если же пользователь ввел некорректное выражение, то программа сообщит об этом:



# Результаты

Была разработана программа-калькулятор, которая вычисляет заданные пользователем арифметические выражения. Она работает с операциями сложения, вычитания, умножения, деления, возведения в степень. Способна работать с числами в десятичной или экспоненциальной форме. Калькулятор имеет два способа вычисления выражений – с помощью постфиксной записи и без нее. Программа протестирована и отлажена. Окончательный вариант программы представлен в виде графического интерфейса Windows Forms.

# Заключение

Таким образом, мы увидели, каким образом можно найти значение выражения за один проход. В заключение можно отметить, что программу можно расширять, добавляя в нее реализацию вычисления других математических функций – cos, sin, log, exp, и т.д.

# Литература

1. Microsoft [электронный ресурс]: // Документация по языку C++

URL: <https://docs.microsoft.com/ru-ru/cpp/cpp/>

1. Р. Лафоре Объектно-ориентированное программирование в C++ – Санкт-Петербург: “Питер”, 2018
2. Википедия – Обратная польская запись  
   <https://ru.wikipedia.org/wiki/Обратная_польская_запись>

***Ссылка на GitHub на репозиторий с проектом:***https://github.com/TechnoWolf96/Lab-Calculator

# Приложение

## TCalculator.h

#pragma once

#include <string>

#include "TStack.h"

#include <math.h>

#define VIRGULE '.' // Какой символ является запятой-разделителем целой и дробной части вещественных чисел

using namespace std;

class TCalculator

{

private:

string expression; // Алгебраическое выражение

string postfix; // Постфиксная запись выражения

TStack<double> numberStack; // Стек с числами

TStack<char> operatorStack; // Стек с операторами и скобками

const string operators = "+-/\*^"; // Строка со всевозможными операторами

void ToPostfix(); // Сохраняет постфиксную запись в строку postfix

int Priority(char oper); // Возвращает приоритет операции

bool CheckExpression(); // Функция проверки введенного выражения

bool IsOperator(char symbol); // Функция проверяет, есть ли оператором переданный символ

void ClearStacks(); // Очистка стеков

public:

TCalculator(string expression = ""); // Конструктор

TCalculator(const TCalculator& other); // Конструктор копирования

~TCalculator() {} // Деструктор

TCalculator& operator=(const TCalculator& other); // Присваивание

void SetExpression(string expression); // Сеттер

string GetExpression() { return expression; } // Геттер

string GetPostfix() { return postfix; } // Получить постфиксную запись

double CalculatePostfix(); // Вычисление значения выражения, используя строку postfix

double Calculate(); // Вычисление выражения без использования заранее подготовленной строки postfix

};

inline void TCalculator::ClearStacks()

{

numberStack.Clear();

operatorStack.Clear();

}

inline bool TCalculator::IsOperator(char symbol)

{

for (int i = 0; i < operators.length(); i++)

{

if (symbol == operators[i]) return true;

}

return false;

}

inline TCalculator::TCalculator(string expression)

{

this->expression = expression;

numberStack = TStack<double>(expression.size());

operatorStack = TStack<char>(expression.size());

if (!CheckExpression()) throw "Wrong expression";

ToPostfix();

}

inline TCalculator::TCalculator(const TCalculator& other)

{

expression = other.expression;

postfix = other.postfix;

numberStack = TStack<double>(other.expression.size());

operatorStack = TStack<char>(other.expression.size());

}

inline TCalculator& TCalculator::operator=(const TCalculator& other)

{

expression = other.expression;

postfix = other.postfix;

numberStack = other.numberStack;

operatorStack = other.operatorStack;

}

inline void TCalculator::SetExpression(string expression)

{

this->expression = expression;

if (!CheckExpression()) throw "Wrong expression";

ToPostfix();

numberStack.~TStack();

operatorStack.~TStack();

numberStack = TStack<double>(expression.size());

operatorStack = TStack<char>(expression.size());

}

inline bool TCalculator::CheckExpression()

{

ClearStacks();

// Проверяем что выражение - это последовательность: число, операция, число, операция и т.д (Скобки игнорируем)

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

{

if (!isdigit(expression[i]) && !IsOperator(expression[i]) && expression[i] != '('

&& expression[i] != ')' && expression[i] != ' ') return false;

if (isdigit(expression[i]))

{

size\_t addIndex;

operatorStack.Push('n');

stod(&expression[i], &addIndex);

if (operatorStack.Count() > 1)

return false;

i += addIndex;

}

if (IsOperator(expression[i]))

{

try { operatorStack.Pop(); }

catch (...) { return false; }

}

if (operatorStack.Count() > 1)

return false;

}

ClearStacks();

// Проверка правильности расстановки скобок

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

{

if (expression[i] == '(') operatorStack.Push('(');

try

{

if (expression[i] == ')') operatorStack.Pop();

}

catch (...)

{

return false;

}

}

if (!operatorStack.Empty()) return false;

return true;

}

inline double TCalculator::CalculatePostfix()

{

ClearStacks();

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

{

if (isdigit(postfix[i])) // Если в постфиксной записи попалось число то:

{

size\_t addIndex;

double number = stod(&postfix[i], &addIndex);

numberStack.Push(number); // Добавляем его в стек с числами

i += addIndex; // Переводим текущий индекс на начало следующей операции/операнда

}

if (IsOperator(postfix[i])) // Встретился знак - нужно произвести данную операцию над двумя последними в стеке числами

{

double second = numberStack.Pop();

double first = numberStack.Pop();

switch (postfix[i])

{

case '+': numberStack.Push(first + second); break;

case '-': numberStack.Push(first - second); break;

case '\*': numberStack.Push(first \* second); break;

case '/': numberStack.Push(first / second); break;

case '^': numberStack.Push(pow(first, second)); break;

}

}

}

return numberStack.Pop();

}

inline double TCalculator::Calculate()

{

ClearStacks();

string infix = '(' + expression + ')';

for (int i = 0; i < infix.length(); i++)

{

if (isdigit(infix[i])) // Если в постфиксной записи попалось число то:

{

size\_t addIndex;

double number = stod(&infix[i], &addIndex);

numberStack.Push(number); // Добавляем его в стек с числами

i += addIndex; // Переводим текущий индекс на начало следующей операции/операнда

}

if (infix[i] == '(') // Встретилась скобка - кладем в стек операций

operatorStack.Push(infix[i]);

if (IsOperator(infix[i])) // Встретился оператор

{

// Попалась менее/равноприоритеная операция - выполняем ту, что на вершине стека

while (Priority(infix[i]) <= Priority(operatorStack.Top()))

{

double second = numberStack.Pop();

double first = numberStack.Pop();

switch (operatorStack.Pop())

{

case '+': numberStack.Push(first + second); break;

case '-': numberStack.Push(first - second); break;

case '\*': numberStack.Push(first \* second); break;

case '/': numberStack.Push(first / second); break;

case '^': numberStack.Push(pow(first, second)); break;

}

}

// И кладем в стек новую операцию

operatorStack.Push(infix[i]);

}

if (infix[i] == ')') // Встретилась закрывающая скобка - выполняем операции в стеке до открывающей скобки

{

while (operatorStack.Top() != '(')

{

double second = numberStack.Pop();

double first = numberStack.Pop();

switch (operatorStack.Pop())

{

case '+': numberStack.Push(first + second); break;

case '-': numberStack.Push(first - second); break;

case '\*': numberStack.Push(first \* second); break;

case '/': numberStack.Push(first / second); break;

case '^': numberStack.Push(pow(first, second)); break;

}

}

operatorStack.Pop(); // Убрали открывающую скобку

}

}

return numberStack.Pop();

}

inline void TCalculator::ToPostfix()

{

ClearStacks();

string infix = "(" + expression + ")";

postfix = "";

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

{

if (isdigit(infix[i])) // Встретили цифру

{

// Считываем полностью вещественное число (Даже в виде 10e5, 10e-5)

while ((isdigit(infix[i]) || infix[i] == VIRGULE || infix[i] == 'e' ||

(i > 0 && infix[i - 1] == 'e' && infix[i] == '-')) && i < infix.size())

{

postfix += infix[i];

i++;

}

postfix += ' ';

}

if (infix[i] == '(') operatorStack.Push(infix[i]);

if (infix[i] == ')')

{

while (operatorStack.Top() != '(')

postfix = postfix + operatorStack.Pop() + ' ';

operatorStack.Pop();

}

if (IsOperator(infix[i]))

{

while (Priority(infix[i]) <= Priority(operatorStack.Top()))

{

postfix = postfix + operatorStack.Pop() + ' ';

}

operatorStack.Push(infix[i]);

}

}

}

inline int TCalculator::Priority(char oper)

{

if (oper == '(') return 0;

if (oper == '+' || oper == '-') return 1;

if (oper == '\*' || oper == '/') return 2;

if (oper == '^') return 3;

}

## TStack.h

#pragma once

// Класс "Стек" со структурой хранения элементов в массиве

template<class T>

class TStack

{

private:

T\* arr; // Массив, в котором хранятся элементы стека

int size; // Размер стека

int pos; // Позиция последнего элемента в стеке

public:

TStack(int \_size = 10); // Конструктор

TStack(const TStack& other); // Конструктор копирования

TStack<T>& operator=(const TStack<T>& other); // Оператор присваивания

~TStack() { delete[] arr; } // Деструктор

int Count() { return pos + 1; } // Количество элементов в стеке

bool Empty(); // Пустой ли стек

bool Full(); // Заполнен ли стек

void Push(T item); // Добавить элемент в стек

T Pop(); // Извлечь последний элемент из стека

T Top(); // Посмотреть последний элемент в стеке

void Clear() { pos = -1; } // Очистить стек

bool operator!=(const TStack<T>& other); // Операторы сравнения

bool operator==(const TStack<T>& other);

};

template<class T>

inline TStack<T>::TStack(int \_size)

{

if (\_size <= 0) throw "Wrong size";

size = \_size;

arr = new T[size];

pos = -1;

}

template<class T>

inline TStack<T>::TStack(const TStack& other)

{

size = other.size;

arr = new T[size];

pos = other.pos;

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

arr[i] = other.arr[i];

}

template<class T>

inline bool TStack<T>::Empty()

{

if (pos == -1) return true;

else return false;

}

template<class T>

inline bool TStack<T>::Full()

{

return pos >= size-1;

}

template<class T>

inline void TStack<T>::Push(T item)

{

if (Full()) throw "Stack owerflow";

arr[++pos] = item;

}

// Извлечь последний элемент из стека

template<class T>

inline T TStack<T>::Pop()

{

if (Empty()) throw "Stack is empty";

T temp = arr[pos];

pos--;

return temp;

}

template<class T>

inline T TStack<T>::Top()

{

if (Empty()) throw "Stack is empty";

return arr[pos];

}

template<class T>

inline TStack<T>& TStack<T>::operator=(const TStack<T>& other)

{

if (size != other.size)

{

delete[] arr;

size = other.size;

arr = new T[size];

}

pos = other.pos;

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

arr[i] = other.arr[i];

return \*this;

}

template<class T>

inline bool TStack<T>::operator!=(const TStack& other)

{

return !operator==(other);

}

template<class T>

inline bool TStack<T>::operator==(const TStack& other)

{

if (pos != other.pos) return false;

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

if (arr[i] != other.arr[i]) return false;

return true; }

## GTest\_TStack.cpp

#include "pch.h"

#include "../Stack/TCalculator.h"

TEST(TStack, Can\_create\_stack\_witn\_positive\_size)

{

EXPECT\_NO\_THROW(TStack<int> stack(5));

}

TEST(TStack, Can\_not\_create\_stack\_witn\_negative\_size)

{

EXPECT\_ANY\_THROW(TStack<int> stack(-1));

}

TEST(TStack, Stack\_is\_empty\_during\_creating)

{

TStack<int> a(1);

EXPECT\_TRUE(a.Empty());

}

TEST(TStack, Check\_working\_equal\_operator)

{

TStack<int> a(3); TStack<int> b;

a.Push(1); a.Push(2); a.Push(3);

b = a;

EXPECT\_TRUE(a == b);

}

TEST(TStack, Stacks\_have\_own\_memory)

{

TStack<int> a(3); TStack<int> b;

a.Push(1); a.Push(2); a.Push(3);

b = a;

b.Pop();

b.Push(4);

EXPECT\_TRUE(a != b);

}

TEST(TStack, Can\_put\_element)

{

TStack<int> a(1);

a.Push(1);

EXPECT\_EQ(1, a.Top());

}

TEST(TStack, Stack\_is\_not\_epmty\_when\_put\_element)

{

TStack<int> a(1);

a.Push(1);

EXPECT\_FALSE(a.Empty());

}

TEST(TStack, Stack\_become\_empty\_when\_extract\_last\_element)

{

TStack<int> a(1);

a.Push(1);

a.Pop();

EXPECT\_TRUE(a.Empty());

}

TEST(TStack, Stack\_is\_full\_when\_stack\_have\_not\_memory)

{

TStack<int> a(2);

a.Push(1); a.Push(2);

EXPECT\_TRUE(a.Full());

}

TEST(TStack, Can\_not\_put\_element\_in\_overflow\_stack)

{

TStack<int> a(2);

a.Push(1); a.Push(2);

EXPECT\_ANY\_THROW(a.Push(3));

}

TEST(TStack, Can\_not\_extract\_element\_when\_stack\_is\_empty)

{

TStack<int> a(3);

EXPECT\_ANY\_THROW(a.Pop());

}

TEST(TStack, Can\_not\_check\_element\_when\_stack\_is\_empty)

{

TStack<int> a(3);

EXPECT\_ANY\_THROW(a.Top());

}

TEST(TStack, Check\_counter)

{

TStack<int> a(5);

a.Push(1); a.Push(2); a.Push(3);

EXPECT\_EQ(3, a.Count());

}

TEST(TStack, Stack\_become\_empty\_when\_worked\_clear)

{

TStack<int> a(5);

a.Push(1); a.Push(2); a.Push(3);

a.Clear();

EXPECT\_TRUE(a.Empty());

}

## GTest\_Calculator.cpp

#include "pch.h"

#include "../Stack/TCalculator.h"

#include <math.h>

TEST(TCalculator, Can\_create\_calculator\_with\_correct\_expression)

{

ASSERT\_NO\_THROW(TCalculator a("()(1)+(2\*(3^4))"));

}

TEST(TCalculator, Can\_not\_create\_calculator\_with\_uncorrect\_expression)

{

ASSERT\_ANY\_THROW(TCalculator a(")(1+2\*3^4"));

ASSERT\_ANY\_THROW(TCalculator a(")4)"));

ASSERT\_ANY\_THROW(TCalculator a("4+2+-4"));

ASSERT\_ANY\_THROW(TCalculator a("5+5,,34"));

ASSERT\_ANY\_THROW(TCalculator a("5/ +24"));

ASSERT\_ANY\_THROW(TCalculator a("5 6/ +24"));

}

TEST(TCalculator, True\_Calculate\_integer\_1)

{

TCalculator a("2+4^9\*9-7/2+3");

EXPECT\_EQ(2 + pow(4.0,9.0) \* 9 - 7.0 / 2.0 + 3 , a.Calculate());

EXPECT\_EQ(2 + pow(4.0, 9.0) \* 9 - 7.0 / 2.0 + 3, a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_integer\_2)

{

TCalculator a("2\*(5\*3-8)\*2^5/6");

EXPECT\_EQ(2 \* (5 \* 3 - 8) \* pow(2.0,5.0) / 6 , a.Calculate());

EXPECT\_EQ(2 \* (5 \* 3 - 8) \* pow(2.0, 5.0) / 6, a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_integer\_3)

{

TCalculator a("(5+9\*8)/((5+6)\*2^4)");

EXPECT\_EQ((5 + 9 \* 8) / ((5 + 6) \* pow(2.0,4.0)), a.Calculate());

EXPECT\_EQ((5 + 9 \* 8) / ((5 + 6) \* pow(2.0, 4.0)), a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_decimal\_1)

{

TCalculator a("5.59-9.2\*(4.12/3.2+8.4)");

EXPECT\_EQ(5.59 - 9.2 \* (4.12 / 3.2 + 8.4), a.Calculate());

EXPECT\_EQ(5.59 - 9.2 \* (4.12 / 3.2 + 8.4), a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_decimal\_2)

{

TCalculator a("8.3/5\*(9-1.4\*7/2)");

EXPECT\_EQ(8.3 / 5 \* (9 - 1.4 \* 7.0 / 2), a.Calculate());

EXPECT\_EQ(8.3 / 5 \* (9 - 1.4 \* 7.0 / 2), a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_decimal\_3)

{

TCalculator a("8.9\*4.15+(85.1/(4.2-1.3))-5.2");

EXPECT\_EQ(8.9 \* 4.15 + (85.1 / (4.2 - 1.3)) - 5.2, a.Calculate());

EXPECT\_EQ(8.9 \* 4.15 + (85.1 / (4.2 - 1.3)) - 5.2, a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_exponential\_1)

{

TCalculator a("5e3 \* (1e-3 + 8e-2) / 8e2 + 24e-4");

EXPECT\_EQ(5e3 \* (1e-3 + 8e-2) / 8e2 + 24e-4, a.Calculate());

EXPECT\_EQ(5e3 \* (1e-3 + 8e-2) / 8e2 + 24e-4, a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_exponential\_2)

{

TCalculator a("12e3 - 10^5 / (8.26 + 71e-4)");

EXPECT\_EQ(12e3 - pow(10.0,5.0) / (8.26 + 71e-4), a.Calculate());

EXPECT\_EQ(12e3 - pow(10.0, 5.0) / (8.26 + 71e-4), a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_exponential\_3)

{

TCalculator a("7.56e3 / 2^7 \* (8.1e4 - 5126e-3)");

EXPECT\_EQ(7.56e3 / pow(2.0,7.0) \* (8.1e4 - 5126e-3), a.Calculate());

EXPECT\_EQ(7.56e3 / pow(2.0, 7.0) \* (8.1e4 - 5126e-3), a.CalculatePostfix());

}

TEST(TCalculator, True\_Calculate\_strange\_expression)

{

TCalculator a("()( 1.25) +94 \* ()() 11e2-2^12");

EXPECT\_EQ((1.25) + 94 \* 11e2 - pow(2.0,12.0), a.Calculate());

EXPECT\_EQ((1.25) + 94 \* 11e2 - pow(2.0, 12.0), a.CalculatePostfix());

}

## CForm.h

#pragma once

#include "../Stack/TCalculator.h"

#include <msclr\marshal\_cppstd.h>

#include <string>

namespace CalculatorForm {

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>

/// Сводка для CForm

/// </summary>

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

{

public:

CForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~CForm()

{

if (components)

{

delete components;

}

}

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

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

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

private: System::Windows::Forms::TextBox^ textBox2;

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

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::TextBox^ textBox3;

protected:

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(CForm::typeid));

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

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

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

this->textBox2 = (gcnew System::Windows::Forms::TextBox());

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

this->label3 = (gcnew System::Windows::Forms::Label());

this->textBox3 = (gcnew System::Windows::Forms::TextBox());

this->SuspendLayout();

//

// textBox1

//

this->textBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->textBox1->Location = System::Drawing::Point(58, 89);

this->textBox1->Multiline = true;

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

this->textBox1->Size = System::Drawing::Size(642, 43);

this->textBox1->TabIndex = 0;

this->textBox1->KeyPress += gcnew System::Windows::Forms::KeyPressEventHandler(this, &CForm::textBox1\_KeyPress);

//

// label1

//

this->label1->AutoSize = true;

this->label1->BackColor = System::Drawing::Color::Transparent;

this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->label1->Location = System::Drawing::Point(227, 42);

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

this->label1->Size = System::Drawing::Size(292, 32);

this->label1->TabIndex = 1;

this->label1->Text = L"Введите выражение:";

//

// button1

//

this->button1->Cursor = System::Windows::Forms::Cursors::Hand;

this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->button1->Location = System::Drawing::Point(273, 186);

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

this->button1->Size = System::Drawing::Size(192, 52);

this->button1->TabIndex = 2;

this->button1->Text = L"Вычислить!";

this->button1->UseVisualStyleBackColor = true;

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

//

// textBox2

//

this->textBox2->BackColor = System::Drawing::SystemColors::Window;

this->textBox2->Enabled = false;

this->textBox2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->textBox2->Location = System::Drawing::Point(217, 335);

this->textBox2->Multiline = true;

this->textBox2->Name = L"textBox2";

this->textBox2->ReadOnly = true;

this->textBox2->Size = System::Drawing::Size(483, 43);

this->textBox2->TabIndex = 3;

//

// label2

//

this->label2->AutoSize = true;

this->label2->BackColor = System::Drawing::Color::Transparent;

this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->label2->Location = System::Drawing::Point(52, 338);

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

this->label2->Size = System::Drawing::Size(159, 32);

this->label2->TabIndex = 4;

this->label2->Text = L"Результат:";

//

// label3

//

this->label3->AutoSize = true;

this->label3->BackColor = System::Drawing::Color::Transparent;

this->label3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->label3->Location = System::Drawing::Point(46, 401);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(165, 64);

this->label3->TabIndex = 6;

this->label3->Text = L"Инфиксная\r\n запись:";

//

// textBox3

//

this->textBox3->BackColor = System::Drawing::SystemColors::Window;

this->textBox3->Enabled = false;

this->textBox3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->textBox3->Location = System::Drawing::Point(217, 411);

this->textBox3->Multiline = true;

this->textBox3->Name = L"textBox3";

this->textBox3->ReadOnly = true;

this->textBox3->Size = System::Drawing::Size(483, 43);

this->textBox3->TabIndex = 5;

//

// CForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

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

this->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"$this.BackgroundImage")));

this->ClientSize = System::Drawing::Size(758, 518);

this->Controls->Add(this->label3);

this->Controls->Add(this->textBox3);

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

this->Controls->Add(this->textBox2);

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

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

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

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedSingle;

this->Name = L"CForm";

this->Text = L"Калькулятор выражений";

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: System::Void textBox1\_KeyPress(System::Object^ sender, System::Windows::Forms::KeyPressEventArgs^ e) {

if ((Char::IsLetter(e->KeyChar) && e->KeyChar != 'e'))

e->Handled = true;

}

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

string expr;

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

try

{

TCalculator a(expr);

textBox2->Text = Convert::ToString(a.Calculate());

textBox3->Text = msclr::interop::marshal\_as<String^>(a.GetPostfix());

}

catch (...)

{

textBox2->Text = "Выражение задано неверно";

textBox3->Text = "";

}

}

};

}