Министерство образования и науки Российской Федерации

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**Разработка структуры данных стек и применение её для решения практической задачи**

**Выполнил**:студент группы 381606-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимакин Н.Е.

Подпись

**Проверил**: к.ф.-м.н., доц.

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

Подпись

Нижний Новгород 2017

**Введение**

Стек – это абстрактный тип данных, представляющий собой список элементов и организованный по принципу «последний пришёл – первым ушёл», то есть добавлять или удалять элементы можно только с одного конца списка. Стек не только часто встречается в повседневной жизни (стопка тарелок или книг), но ещё с помощью него решаются многие прикладные задачи, например в теории графов. Одна из таких задач и будет рассмотрена как пример использования этого типа данных.

**Постановка задачи**

Реализовать тип данных стек как шаблонный класс, написать к нему тесты, подтверждающие его работоспособность, и тестовую программу – калькулятор, в двух вариантах: в виде консольного приложения и с графическим интерфейсом.

**Описание структуры программы**

Сам стек будет реализован как класс TStack в одном .h-файле, и для калькулятора понадобится написать отдельный класс TCalculator, который будет проводить все нужные операции для вычисления результата и использовать для этого поля типа TStack.

Программа содержит в себе 3 проекта:

1. Stack-lab – реализация классов TStack и TCalculator, тестовая программа (консольное приложение)
   * stack.h – объявление и реализация класса TStack
   * TCalculator.h – объявление класса TCalculator
   * TCalculator.cpp – реализация класса TCalculator
   * main.cpp – реализация тестовой программы
2. Test – тесты для стека и калькулятора
   * gtest-all.cc – библиотека с реализацией тестов
   * TCalculator.cpp (добавлен для работы с TCalculator)
   * test\_calculator.cpp – тесты для класса TCalculator
   * test\_main.cpp – файл, запускающий все тесты
   * test\_stack.cpp – тесты для класса TStack
3. Visual calculator – тестовая программа (приложение с графическим интерфейсом)
   * MyForm.h – объявление и реализация формы для графического приложения
   * MyForm.cpp – файл для запуска приложения
   * TCalculator.cpp (добавлен для работы с TCalculator)

T - шаблон

**Класс TStack**

Поля:

T\* arr - память для хранения элементов

int size - текущий размер

int maxsize - максимальный размер

Методы:

TStack(int \_maxsize = 10) – конструктор по умолчанию

TStack(const TStack& st) – конструктор копирования

~TStack() - деструктор

TStack& operator=(const TStack& st) – перегрузка оператора присваивания

int isfull() – проверка на полноту

int isempty() – проверка на пустоту

T top() – возвращение вершины стека

T pop() – удаление элемента

void push(const T& el) – добавление элемента

int getsize() – возвращение текущего размера

int getmaxsize() – возвращение максимального размера

int operator==(const TStack& st) const – перегрузка операции сравнения

int operator!=(const TStack& st) const – перегрузка операции не равно

void clear() – очистка стека

**Класс TCalculator**

Выражение, которое необходимо посчитать, будет поступать в программу в виде строки. Чтобы эту строку символов превратить в последовательность математических операций, а затем и в вычисление результата, воспользуемся постфиксной записью, то есть вместо привычной для нас инфиксной строки <операнд 1> <оператор> <операнд 2> будем использовать такую: <операнд 1> <операнд 2> <оператор>. Например, строка “2+4” превратится в “2 4+”, или “5-3\*6” – “5 3 6\*-“, или “(1.1+2.7/3)^2” – “1.1 2.7 3/+ 2 ^”. Это позволит нам реализовать калькулятор именно с помощью стека.

Поля:

string infix – инфиксная строка

string postfix – постфиксная строка

TStack<char> stc – стек для знаков

TStack<double> StD – стек для чисел

Методы:

TCalculator(string \_infix = "") – конструктор по умолчанию

void setinfix(string \_infix) – задание инфиксной строки

string getinfix() – получение инфиксной строки

string getpostfix() – получение постфиксной строки

int check() – проверка на корректность инфиксной строки

int priority(char sym) - определение приоритета символа

void topostfix() – перевод инфиксной строки в постфиксную

double calc() – подсчёт результата

**Описание алгоритмов**

**Класс TStack**

TStack(int \_maxsize = 10) {если \_maxsize <= 0, то выбросить исключение. Максимальному размеру присвоить \_maxsize, текущему размеру – 0. Выделить память объёмом максимального размера}

TStack(const TStack& st) {максимальному размеру присвоить максимальный размер st. Текущему размеру присвоить текущий размер st. Выделить память объёмом максимального размера. Каждому элементу присвоить значение соответствующего из st}

~TStack() {освободить память}

TStack& operator=(const TStack& st) {если this не указывает на st, то если максимальные размеры не равны, то максимальному размеру присвоить максимальный размер st, освободить память, выделить новую; текущему размеру присвоить текущий размер st, каждому элементу присвоить значение соответствующего из st. Вернуть \*this}

int isfull() {если текущий размер совпадает с максимальным, то вернуть 1, иначе – 0}

int isempty() {если текущий размер равен 0, то вернуть – 1, иначе – 0}

T top() {если стек пустой, то выбросить исключение. Вернуть элемент с номером (текущий размер – 1)}

T pop() {если стек пустой, то выбросить исключение. Уменьшить текущий размер на 1. Вернуть элемент с номером (текущий размер)}

void push(const T& el) {если стек полный, то выбросить исключение. Записать в элемент с номером (текущий размер) значение el. Увеличить размер на 1}

int getsize() {вернуть текущий размер}

int getmaxsize() {вернуть максимальный размер}

int operator==(const TStack& st) const {если максимальные размеры не совпадают, вернуть 0. Если текущие размеры не совпадают, вернуть 0. Если хотя бы один элемент не совпадает, вернуть 0. Вернуть 1}

int operator!=(const TStack& st) const {вернуть отрицание от проверки на равенство \*this и st}

void clear() {присвоить текущему размеру 0}

**Класс TCalculator**

TCalculator(string \_infix = "") {вызвать конструкторы для знакового и числового стеков с параметром (\_infix + 1). Инфиксной строке присвоить \_infix}

void setinfix(string \_infix) {инфиксной строке присвоить \_infix}

string getinfix() {вернуть инфиксную строку}

string getpostfix() {вернуть постфиксную строку}

int check() {очистить стек. В цикле от 0 до длины инфиксной строки: если i-ый символ инфиксной строки ‘(‘, то добавить ‘(‘ в знаковый стек; если i-ый элемент ‘)‘, то, если стек пустой – вернуть 0, удалить элемент из стека. Если стек пустой, то вернуть 1, иначе – 0}

int priority(char sym) {если sym это ‘(’, вернуть 0. Если sym это ‘+’ или ‘-‘, вернуть 1. Если sym это ‘\*’ или ‘/’, вернуть 2. Если sym это ‘^’, вернуть 3. Иначе вернуть -1}

void topostfix() {

если инфиксная строка корректна{

постфиксу присвоить пустую строку

очистить знаковый стек

создать строку-буфер, такую же как инфикс только со скобками в начале и в конце

цикл от 0 до длины буфера{

если i-ый символ буфера ‘(’{

добавить ‘(’ в стек

}

если i-ый символ буфера – цифра или ‘.’{

прибавить i-ый символ к постфиксу

}

если i-ый символ буфера ‘)’{

создать временный символ, равный вершине стека (применить pop())

пока временный символ не равен ‘)’{

прибавить к постфиксу временный символ

извлечь элемент из стека во временный символ

}

}

если i-ый символ буфера ‘+’, ‘-’, ‘\*’, ‘/’ или ‘^’{

прибавить к постфиксу пробел

пока приоритет i-ого символа меньше или равен приоритету вершины стека{

прибавить к постфиксу вершину стека (применить pop())

}

}

}

}

иначе{

выбросить исключение

}

}

double calc() {

перевести инфиксную строку в постфиксную

очистить числовой стек

в цикле от 0 до длины постфикса{

если i-ый элемент постфикса ‘+’, ‘-’, ‘\*’, ‘/’ или ‘^’{

создать два числа – операнд1 и операнд2

операнду2 присвоить вершину стека (pop())

операнду1 присвоить вершину стека (pop())

если i-ый символ:

‘+’: добавить в стек сумму операндов

‘-’: добавить в стек разность операндов

‘\*’: добавить в стек произведение операндов

‘/’: добавить в стек частное операндов

‘^’: добавить в стек оп1 в степени оп2

}

если i-ый элемент постфикса – цифра или ‘.’{

создать указатель на символ (tmp) – указатель на конец строки

создать вещественный результат = strtod(&postfix[i], &tmp);

создать целочисленную переменную di = tmp - &postfix[i];

i += di -1;

результат добавить в стек

}

}

создать результат равный вершине стека

если стек не пустой, то выбросить исключение

вернуть результат

}

**Консольное приложение**

Этот вариант тестовой программы содержит только одну функцию – main

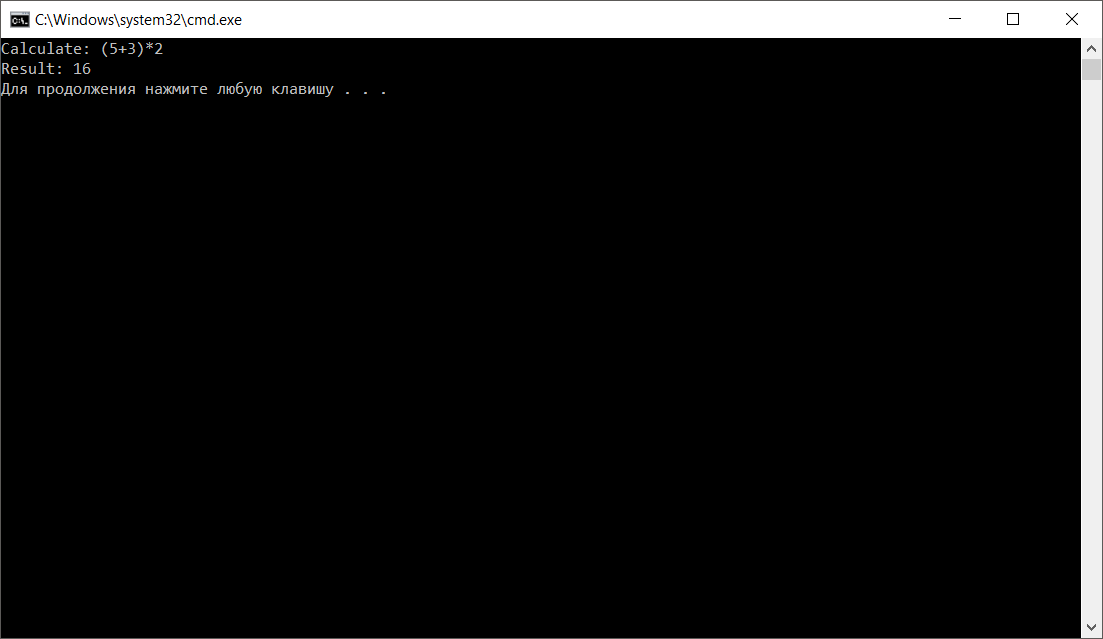
int main() {создать и ввести с клавиатуры строку. Создать объект класса TCalculator с параметром строка под результат. Вывести на экран результат вычисления строки. Если отлавливается исключение, то на экран выводится сообщение об ошибке}

**Приложение с графическим интерфейсом**

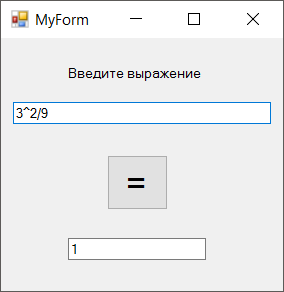
На форме расположена одна кнопка, label и 2 textBox. В textBox1 вводится строка, создаётся объект класса TCalculator с параметром строка под результат. По нажатию кнопки или клавиши Enter в textBox2 выводится результат вычислений или, если было поймано исключение, сообщение об ошибке.

**Результаты**

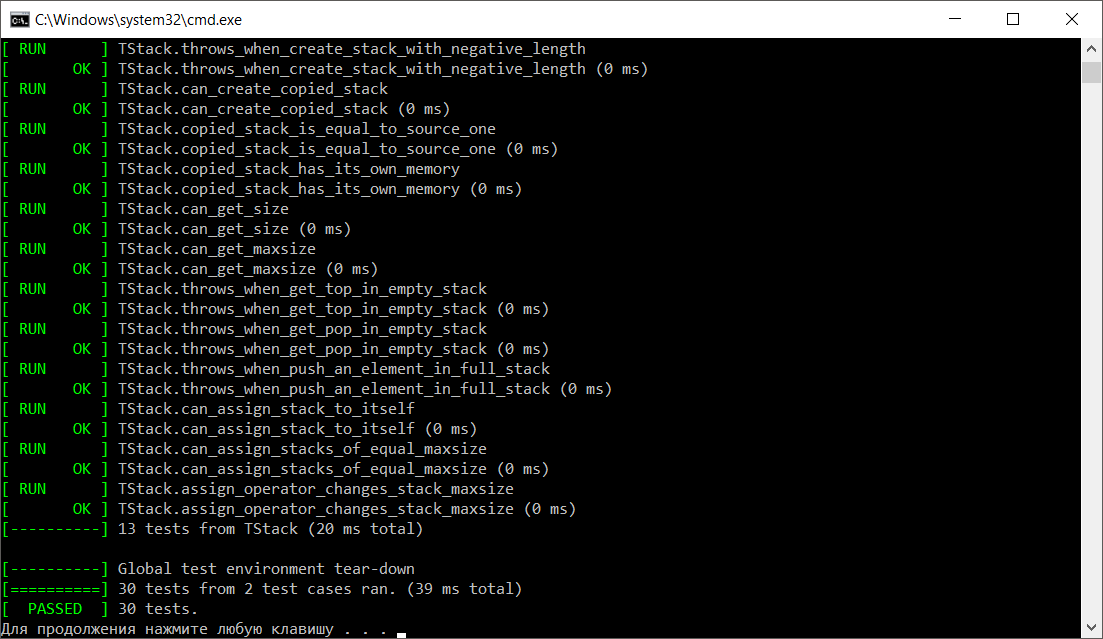
Консольное приложение:



Приложение с графическим интерфейсом:



Тесты:



**Вывод**

Оба варианта тестовых программ работают корректно, все тесты пройдены, все задачи, поставленные ранее, выполнены, а значит, разработку структуры данных стек и её применение в практической задаче можно считать успешными.

**Литература**

Брайан Керниган, Деннис Ритчи «Язык программирования Си»

Брюс Эккель «Философия С++. Введение в стандартный С++»

Стивен Прата «Язык программирования С++. Лекции и упражнения»

**Приложение**

**stack.h**

#pragma once

#include <iostream>

using namespace std;

template <class T>

class TStack {

T\* arr;

int size;

int maxsize;

public:

TStack(int \_maxsize = 10) {

if (\_maxsize <= 0)

throw \_maxsize;

maxsize = \_maxsize;

size = 0;

arr = new T[maxsize];

}

TStack(const TStack& st) {

maxsize = st.maxsize;

size = st.size;

arr = new T[maxsize];

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

arr[i] = st.arr[i];

}

}

~TStack() {

delete[] arr;

}

TStack& operator=(const TStack& st) {

if (this != &st) {

if (maxsize != st.maxsize) {

maxsize = st.maxsize;

delete[] arr;

arr = new T[maxsize];

}

size = st.size;

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

arr[i] = st.arr[i];

}

}

return \*this;

}

int isfull() {

if (size == maxsize) return 1;

return 0;

}

int isempty() {

if (size == NULL) return 1;

return 0;

}

T top() {

if (isempty()) throw "top error";

return arr[size - 1];

}

T pop() {

if (isempty()) throw "pop error";

size--;

return arr[size];

}

void push(const T& el) {

if (isfull()) throw "push error";

arr[size] = el;

size++;

}

int getsize() {

return size;

}

int getmaxsize() {

return maxsize;

}

int operator==(const TStack& st) const {

if (maxsize != st.maxsize)

return 0;

if (size != st.size)

return 0;

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

if (arr[i] != st.arr[i])

return 0;

}

return 1;

}

int operator!=(const TStack& st) const {

return !(\*this == st);

}

void clear() {

size = 0;

}

};

**TCalculator.h**

#pragma once

#include "stack.h"

#include <string>

class TCalculator {

string infix;

string postfix;

TStack<char> stc;

TStack<double> StD;

public:

TCalculator(string \_infix = "") : stc(\_infix.size() + 1), StD(\_infix.size() + 1) {

infix = \_infix;

}

void setinfix(string \_infix) {

infix = \_infix;

}

string getinfix() {

return infix;

}

string getpostfix() {

return postfix;

}

int check();

int priority(char sym);

void topostfix();

double calc();

};

**TCalculator.cpp**

#include "TCalculator.h"

int TCalculator::check() {

stc.clear();

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

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

stc.push('(');

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

if (stc.isempty())

return 0;

stc.pop();

}

}

if (stc.isempty())

return 1;

return 0;

}

int TCalculator::priority(char sym) {

switch (sym) {

case '(': return 0;

case '+': return 1;

case '-': return 1;

case '\*': return 2;

case '/': return 2;

case '^': return 3;

default: return -1;

}

}

void TCalculator::topostfix() {

if (check()) {

postfix = "";

stc.clear();

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

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

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

stc.push('(');

if ((buf[i] <= '9' && buf[i] >= '0') || buf[i] == '.')

postfix += buf[i];

if (buf[i] == ')') {

char el = stc.pop();

while (el != '(') {

postfix += el;

el = stc.pop();

}

}

if (buf[i] == '+' || buf[i] == '-' || buf[i] == '\*' || buf[i] == '/' || buf[i] == '^') {

postfix += " ";

while (priority(buf[i]) <= priority(stc.top())) {

postfix += stc.pop();

}

stc.push(buf[i]);

}

}

}

else

throw - 3;

}

double TCalculator::calc() {

topostfix();

StD.clear();

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

if (postfix[i] == '+' || postfix[i] == '-' || postfix[i] == '\*' || postfix[i] == '/' || postfix[i] == '^') {

double op1, op2;

op2 = StD.pop();

op1 = StD.pop();

switch (postfix[i]) {

case '+': StD.push(op1 + op2); break;

case '-': StD.push(op1 - op2); break;

case '\*': StD.push(op1 \* op2); break;

case '/': StD.push(op1 / op2); break;

case '^': StD.push(pow(op1, op2)); break;

}

}

if ((postfix[i] >= '0' && postfix[i] <= '9') || postfix[i] == '.') {

char\* tmp;

double res = strtod(&postfix[i], &tmp);

int di = tmp - &postfix[i];

i += di - 1;

StD.push(res);

}

}

double result = StD.pop();

if (!StD.isempty())

throw - 2;

return result;

}

**main.cpp**

#include "Stack.h"

#include "TCalculator.h"

int main() {

cout << "Calculate: ";

string str;

cin >> str;

try {

TCalculator result(str);

cout << "Result: " << result.calc() << endl;

}

catch (...) {

cout << "Input error\n";

}

return 0;

}

**test\_stack.cpp**

#include "..\Stack-lab\Stack.h"

#include "gtest.h"

TEST(TStack, TStack\_create\_stack\_with\_positive\_length){

ASSERT\_NO\_THROW(TStack<int> st(3));

}

TEST(TStack, throws\_when\_create\_stack\_with\_negative\_length){

ASSERT\_ANY\_THROW(TStack<int> st(-3));

}

TEST(TStack, can\_create\_copied\_stack)

{

TStack<int> st(5);

ASSERT\_NO\_THROW(TStack<int> st1(st));

}

TEST(TStack, copied\_stack\_is\_equal\_to\_source\_one)

{

int i;

TStack<int> a(3), b;

for (i = 0; i < 3; i++) {

a.push(i);

}

b = a;

EXPECT\_EQ(a, b);

}

TEST(TStack, copied\_stack\_has\_its\_own\_memory)

{

TStack<int> a(3), b;

for (int i = 0; i < a.getmaxsize(); i++) {

a.push(i);

}

b = a;

int tmp = b.pop();

b.push(5);

EXPECT\_NE(a, b);

}

TEST(TStack, can\_get\_size){

TStack<int> st(5);

for (int i = 0; i < 3; i++) {

st.push(i);

}

EXPECT\_EQ(3, st.getsize());

}

TEST(TStack, can\_get\_maxsize){

TStack<int> st(5);

EXPECT\_EQ(5, st.getmaxsize());

}

TEST(TStack, throws\_when\_get\_top\_in\_empty\_stack){

TStack<int> st(3);

ASSERT\_ANY\_THROW(st.top());

}

TEST(TStack, throws\_when\_get\_pop\_in\_empty\_stack){

TStack<int> st(3);

ASSERT\_ANY\_THROW(st.pop());

}

TEST(TStack, throws\_when\_push\_an\_element\_in\_full\_stack) {

TStack<int> st(3);

for (int i = 0; i < 3; i++) {

st.push(i);

}

ASSERT\_ANY\_THROW(st.push(3));

}

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

{

TStack<int> st(3);

ASSERT\_NO\_THROW(st = st);

}

TEST(TStack, can\_assign\_stacks\_of\_equal\_maxsize)

{

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

b = a;

EXPECT\_EQ(a, b);

}

TEST(TStack, assign\_operator\_changes\_stack\_maxsize)

{

TStack<int> a(3), b(5);

b = a;

EXPECT\_EQ(3, b.getmaxsize());

}

**test\_calculator.cpp**

#include "..\Stack-lab\TCalculator.h"

#include "gtest.h"

TEST(TCalculator, can\_create\_object) {

ASSERT\_NO\_THROW(TCalculator a("2+2"));

}

TEST(TCalculator, false\_check\_when\_opening\_brackets\_more\_then\_closing\_breackets) {

TCalculator a("((2+2\*3)");

EXPECT\_EQ(0, a.check());

}

TEST(TCalculator, false\_check\_when\_closing\_brackets\_more\_then\_opening\_breackets) {

TCalculator a("(2+2)\*3)");

EXPECT\_EQ(0, a.check());

}

TEST(TCalculator, false\_check\_when\_the\_first\_symbol\_is\_closing\_breacket) {

TCalculator a(")((2+2\*3)");

EXPECT\_EQ(0, a.check());

}

TEST(TCalculator, false\_check\_when\_the\_last\_symbol\_is\_opening\_breacket) {

TCalculator a("(2+2\*3))(");

EXPECT\_EQ(0, a.check());

}

TEST(TCalculator, throw\_when\_try\_topostfix\_incorrect\_string) {

TCalculator a(")2+3)(");

ASSERT\_ANY\_THROW(a.topostfix());

}

TEST(TCalculator, throw\_when\_try\_to\_calculate\_incorrect\_string) {

TCalculator a(")2+3)(");

ASSERT\_ANY\_THROW(a.calc());

}

TEST(TCalculator, can\_get\_infix) {

TCalculator a("2+2");

EXPECT\_EQ("2+2", a.getinfix());

}

TEST(TCalculator, can\_set\_infix) {

TCalculator a("");

a.setinfix("2+2");

EXPECT\_EQ("2+2", a.getinfix());

}

TEST(TCalculator, can\_get\_postfix) {

TCalculator a("(2+3)\*4");

a.topostfix();

EXPECT\_EQ("2 3+ 4\*", a.getpostfix());

}

TEST(TCalculator, can\_add\_numbers) {

TCalculator a("2+2");

EXPECT\_EQ(4, a.calc());

}

TEST(TCalculator, can\_substract\_numbers) {

TCalculator a("5-2");

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

}

TEST(TCalculator, can\_multiply\_numbers) {

TCalculator a("2\*7");

EXPECT\_EQ(14, a.calc());

}

TEST(TCalculator, can\_divide\_numbers) {

TCalculator a("15/3");

EXPECT\_EQ(5, a.calc());

}

TEST(TCalculator, can\_pow\_numbers) {

TCalculator a("3^3");

EXPECT\_EQ(27, a.calc());

}

TEST(TCalculator, can\_work\_with\_double\_numbers) {

TCalculator a("((2.6+7.8)^0.3-4.5/8.7)\*5.55");

EXPECT\_EQ((pow(2.6 + 7.8, 0.3) - 4.5 / 8.7)\*5.55, a.calc());

}

TEST(TCalculator, can\_calculate\_a\_large\_string) {

TCalculator a("9+(5\*45-(7+7^4)\*5/(7\*2+21/254)+4\*9^6)\*(7+34+9\*(9+3+4^7/8-6+562-5)\*6)/10^7");

double result = 9 + (5 \* 45 - (7 + pow(7, 4)) \* 5 / (7 \* 2 + 21.0 / 254) + 4 \* pow(9, 6))\*(7 + 34 + 9 \* (9 + 3 + pow(4, 7) / 8 - 6 + 562 - 5) \* 6) / pow(10, 7);

cout.precision(100);

cout << a.calc() << endl << result << endl;

EXPECT\_EQ(result, a.calc());

//EXPECT\_EQ(201.064, a.calc());

}

**MyForm.h**

#pragma once

#include "..\Stack-lab\Stack.h"

#include "..\Stack-lab\TCalculator.h"

#include <msclr\marshal\_cppstd.h>

namespace Visual\_calculator {

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();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MyForm()

{

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;

protected:

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

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

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

this->SuspendLayout();

//

// textBox1

//

this->textBox1->Location = System::Drawing::Point(12, 64);

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

this->textBox1->Size = System::Drawing::Size(258, 22);

this->textBox1->TabIndex = 0;

this->textBox1->TextChanged += gcnew System::EventHandler(this, &MyForm::textBox1\_TextChanged);

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

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(64, 27);

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

this->label1->Size = System::Drawing::Size(141, 17);

this->label1->TabIndex = 1;

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

//

// button1

//

this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 20, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->button1->Location = System::Drawing::Point(106, 117);

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

this->button1->Size = System::Drawing::Size(61, 55);

this->button1->TabIndex = 2;

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

this->button1->UseVisualStyleBackColor = true;

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

//

// textBox2

//

this->textBox2->Location = System::Drawing::Point(67, 200);

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

this->textBox2->Size = System::Drawing::Size(138, 22);

this->textBox2->TabIndex = 3;

//

// MyForm

//

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

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

this->ClientSize = System::Drawing::Size(282, 253);

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

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

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

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

this->Name = L"MyForm";

this->Text = L"MyForm";

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: System::Void textBox1\_TextChanged(System::Object^ sender, System::EventArgs^ e) {

}

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

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

TCalculator a(\_infix);

try {

double res = a.calc();

textBox2->Text = System::Convert::ToString(res);

}

catch (...) {

textBox2->Text = "Input error";

}

}

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

if (e->KeyChar == 13) {

button1\_Click(sender, e);

}

}

};

}

**MyForm.cpp**

#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

//[STAThread]

//void Main(array<String^>^ args)

/\*void Main(array<String^>^ args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Visual\_calculator::MyForm form;

Application::Run(%form);

}\*/

#include <Windows.h>

using namespace Visual\_calculator;

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MyForm);

return 0;

}