Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе №3**

**«Вычисление арифметических выражений (стеки)»**

**Выполнил**:студент группы 381703-2

Родионов Денис Михайлович

**Проверил**:

Доцент кафедры МОСТ, к.т.н.

Сысоев А.B.

Нижний Новгород

2018

Содержание

[Введение 3](#_Toc270962758)

[Постановка задачи 4](#_Toc270962759)

[Руководство пользователя 5](#_Toc270962760)

[Руководство программиста 7](#_Toc270962761)

[Описание структур программы 7](#_Toc270962762)

[Описание алгоритмов 7](#_Toc270962763)

[Заключение 11](#_Toc270962765)

[Литература 12](#_Toc270962766)

[Приложения 1](#_Toc270962767)3

# 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.

# Постановка задачи

Требуется разработать вычислитель арифметических выражений. Ставится задача реализации программ, обеспечивающих поддержку стеков, и разработки программных средств, производящих обработку арифметических выражений, включая проверку правильности записи выражения, перевод в постфиксную форму и вычисление результата.

Проверка записи выражения на корректность состоит из:

1) Правильности расстановки скобок,

2) Правильности расстановки знаков операций.

Перевод в постфиксную форму производится только для корректных выражений, а вычисление – для корректных выражений, содержащих числовые операнды, переменные произвольного названия и допустимые знаки операций: сложения, вычитания, умножения и деления.

# Руководство пользователя

Данная программа написана в среде разработки Microsoft Visual Studio 2017 на языке С++.

На первом этапе работы программы пользователю необходимо ввести арифметическое (инфиксное) выражение. Запрещается использование пробелов и разрешается использование вещественных чисел и различных переменных

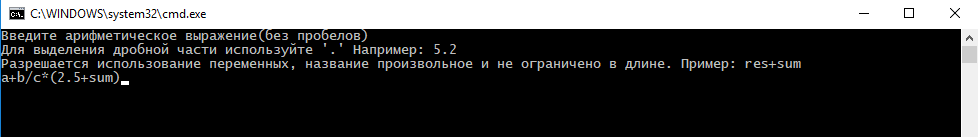


Рис.1. Старт программы

В результате получим выражение, с которым в дальнейшем будем работать



Рис.2. Введенное пользователем выражение

После этого пользователю необходимо проинициализировать значения всех заданных переменных поочередно

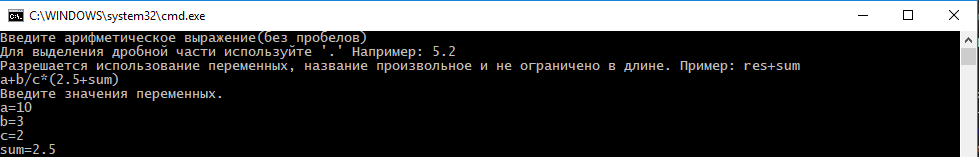


Рис.3. Инициализация переменных

Затем пользователю будут показаны результат вычисления и постфиксная форма заданного выражения, после чего предлагается выбрать одно из следующих действий:

1. Ввести другое арифметическое выражение,
2. Ввести другие значения переменных (если они присутствуют),
3. Выйти из программы.

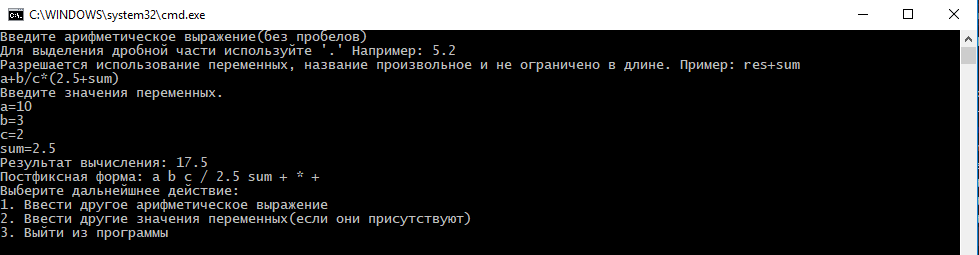


Рис.4.Вывод результата.

# Руководство программиста

В программе реализованы два класса под названием «Стек» и «Постфикс».

**Класс «Стек» - шаблонный, имеющий три приватных поля:**

1. pMem: указатель на объект шаблонного типа, отвечающий за хранение элементов стека,
2. size: переменная типа int, отвечающая за размер стека,
3. top: переменная типа int, отвечающая за индекс последнего элемента в стеке (инициализирована значением -1, если стек пуст).

Помимо этого, в шаблонном классе «Стек» имеются следующие методы:

1. Все необходимые служебные методы,
2. IsEmpty: метод, который проверяет стек на пустоту и возвращает тип bool,
3. IsFull: метод, который проверяет стек на полноту и возвращает тип bool,
4. Clear: метод, который очищает стек,
5. Erase: метод, который удаляет последний элемент в стеке,
6. Pop: метод, который возвращает последний элемент в стеке и удаляет его,
7. PopWithoutDelete: метод, который возвращает последний элемент в стеке при этом не удаляя его,
8. Push: метод, который записывает элемент в стек,
9. Length: метод, который возвращает количество элементов в стеке на данный момент.

**Класс «Постфикс» имеет следующие приватные поля:**

1. infix: переменная типа string, хранит в себе исходное арифметическое выражение,
2. postfix: переменная типа string, хранит в себе преобразованное выражение в постфиксную форму,
3. value: указатель на тип double, хранит в себе соответствующие значения переменных в арифметическом выражении,
4. stackOperation: объект класса «Стек», нужен для преобразования в постфикс и проверки на корректность,
5. stackArguments: объект класса «Стек», нужен для хранения аргументов,
6. stackValue: объект класса «Стек», нужен для хранения соответствующих значений,
7. stackValuePostfix: объект класса «Стек», нужен для высчитывания результата.

Помимо этого, в шаблонном классе «Постфикс» имеются следующие методы:

1. Все необходимые служебные методы,
2. GetInfix и GetPostfix: методы, возвращающие поля infix и postfix соответственно,
3. ChangeInfix: метод, изменяющий значение переменной infix (задание нового арифметического выражения),
4. IsCorrect: метод, проверяющий арифметическое выражение на корректность,
5. ToPostfix: метод, получающий из infix значение postfix,
6. ReadArguments: метод, считающий и инициализирующий всевозможные переменные в арифметическом выражении,
7. Calculate: метод, вычисляющий результат выражения.

Кроме того, программа содержит в себе следующие вспомогательные функции:

1. IsOperation: функция, которая проверяет переданный символ на операцию,
2. IsOperationForCorrect: фнкция, которая проверяет символ на операцию, не учитывая скобок (для проверки на корректность),
3. Priority: функция, возвращающая приоритет переданной операции,
4. Result: функция, вычисляющая отдельное арифметическое действие над двумя значениями с помощью переданной операции,
5. IsNumber: функция, которая проверяет переданный символ на число.

## Описание структуры программы

Программа содержит три основных проекта:

1. base,
2. base\_test,
3. gtest.

Проект base содержит модули “stack.h” с объявлением и реализацией класса «Стек», “postfix.h” с объявлением класса «Постфикс», “postfix.cpp” с реализацией методов класса «Постфикс» и с вспомогательными функциями и “sample\_postfix.cpp” для составления пользовательского меню.

Проект base\_test содержит набор необходимых тестов Google Test, проверяющих правильность реализации основных классов.

Проект gtest содержит необходимую структуру для работы тестов Google Test.

В целом, программа построена на использовании интуитивно понятного пользователю меню. В основной части главной функции программы main создается объект типа «Постфикс», проверяется корректность арифметического выражения, считываются аргументы. Главная функция представляет собой оператор множественного выбора swich. В зависимости от выбора пользователем действия, очередной раз повторяется то или иное действие.

## Описание алгоритмов

**Опишем алгоритмы реализации методов класса «Стек»:**

1. IsEmpty: метод, который проверяет стек на пустоту и возвращает тип bool

*Метод сравнивает значение поля класса top со значением -1.*

1. IsFull: метод, который проверяет стек на полноту и возвращает тип bool

*Метод сравнивает значение поля класса top со значением (MaxStackSize – 1).*

1. Clear: метод, который очищает стек

*Метод присваивает переменной top значение -1.*

1. Erase: метод, который удаляет последний элемент в стеке

*Если стек не пуст, метод уменьшает значение переменной top на единицу.*

1. Pop: метод, который возвращает последний элемент в стеке и удаляет его

*Метод возвращает значение крайнего элемента и уменьшает значение переменной top на единицу.*

1. PopWithoutDelete: метод, который возвращает последний элемент в стеке при этом не удаляя его

*Метод возвращает значение крайнего элемента и не изменяет значение переменной top.*

1. Push: метод, который записывает элемент в стек

*Метод записывает элемент в стек и увеличивает значение переменной top на единицу.*

1. Length: метод, который возвращает количество элементов в стеке на данный момент.

*Метод возвращает значение (top + 1).*

**Опишем алгоритмы реализации методов класса «Постфикс»:**

1. GetInfix и GetPostfix: методы, возвращающие поля infix и postfix соответственно

*Методы возвращают значения соответствующих значения переменных.*

1. ChangeInfix: метод, изменяющий значение переменной infix (задание нового арифметического выражения

*Метод, который принимает «новую» инфиксную форму, присваивает это значение полю infix и удаляет «старое» значение постфиксной формы.*

1. IsCorrect: метод, проверяющий арифметическое выражение на корректность

*Метод, который с помощью цикла for проверяет правильность расстановок операций (любое арифметическое выражение не должно начинаться и закачиваться операциями: +, -, \*, / и те же самые операции не могут следовать в выражении сразу друг за другом). Если операции расставлены корректно, следует проверка на правильность расстановок скобок. Идея этого алгоритма следующая:*

* *Выражение просматривается посимвольно слева направо. Все символы, кроме скобок, игнорируются (т.е. просто производится переход к просмотру следующего символа).*
* *Если очередной символ – открывающая скобка, то её порядковый номер помещается в стек.*
* *Если очередной символ – закрывающая скобка, то производится выталкивание из стека номера открывающей скобки и запись этого номера в паре с номером закрывающей скобки в результирующую таблицу.*
* *Если в этой ситуации стек оказывается пустым, то вместо номера открывающей скобки записывается 0, а счетчик ошибок увеличивается на единицу.*
* *Если после просмотра всего выражения стек оказывается не пустым, то выталкиваются все оставшиеся номера открывающих скобок и записываются в результирующий массив в паре с 0 на месте номера закрывающей скобки, счетчик ошибок каждый раз увеличивается на единицу.*

1. ToPostfix: метод, получающий из infix значение postfix

*Входная строка просматривается посимвольно слева направо до достижения конца строки. Операндами будем считать любую последовательность символов входной строки, не совпадающую со знаками определённых в таблице операций. Операнды по мере их появления переписываются в выходную строку. При появлении во входной строке операции, происходит вычисление приоритета данной операции. Знак данной операции помещается в стек, если:*

* *Приоритет операции равен 0 (это « ( » ),*
* *Приоритет операции строго больше приоритета операции, лежащей на вершине стека,*
* *Стек пуст.*

*В противном случае из стека извлекаются все знаки операций с приоритетом больше или равным приоритету текущей операции. Они переписываются в выходную строку, после чего знак текущей операции помещается в стек. Имеется особенность в обработке закрывающей скобки. Появление закрывающей скобки во входной строке приводит к выталкиванию и записи в выходную строку всех знаков операций до появления открывающей скобки. Открывающая скобка из стека выталкивается, но в выходную строку не записывается. Таким образом, ни открывающая, ни закрывающая скобки в выходную строку не попадают. После просмотра всей входной строки происходит последовательное извлечение всех элементов стека с одновременной записью знаков операций, извлекаемых из стека, в выходную строку.*

1. ReadArguments: метод, считающий и инициализирующий всевозможные переменные в арифметическом выражении

*Метод с помощью цикла for считывает всевозможные переменные в арифметическом выражении, после чего просит пользователя ввести их значения, сохраняя все при этом в указатель value.*

1. Calculate: метод, вычисляющий результат выражения

*Выражение просматривается посимвольно слева направо. При обнаружении операнда производится перевод его в числовую форму и помещение в стек (если операнд не является числом, то вычисление прекращается с выдачей сообщения об ошибке.) При обнаружении знака операции происходит извлечение из стека двух значений, которые рассматриваются как операнд2 и операнд1 соответственно, и над ними производится обрабатываемая операция. Результат этой операции помещается в стек. По окончании просмотра всего выражения из стека извлекается окончательный результат.*

# Заключение

Текущая лабораторная работа познакомила нас с такой схемой запоминания информации, как стек, без которой было бы намного сложнее реализовать данную программу. Стек позволяет управлять памятью наиболее эффективным образом, что значительно упрощает некоторые алгоритмы.

# Литература

1. Бьерн Страуструп. «Язык программирования С++»,
2. Роберт Лафоре. «Объектно-ориентированное программирование в С++»,
3. Герберт Шилдт. «C++. Руководство для начинающих»,
4. Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. «Лабораторный практикум. Учебно-методическое пособие».

# 

# Приложение

**Модуль “stack.h”**

#ifndef \_\_STACK\_H\_\_

#define \_\_STACK\_H\_\_

const int MaxStackSize = 100;

template <class T>

class TStack

{

T \*pMem;

int size;

int top;

public:

TStack(int \_size = MaxStackSize)

{

if ((\_size < 1) || (\_size > MaxStackSize))

throw "data is not correct";

size = \_size;

top = -1;

pMem = new T[size];

}

~TStack()

{

delete[] pMem;

}

bool IsEmpty()

{

return top == -1;

}

bool IsFull()

{

return top == size - 1;

}

void Clear()//очистить

{

top = -1;

}

void Erase()//удалить последний

{

if (!IsEmpty())

top--;

}

T Pop()

{

return pMem[top--];

}

T PopWithoutDelete()//посмотреть без удаления

{

return pMem[top];

}

void Push(T v)

{

pMem[++top] = v;

}

int Length() //элементов в стэке в момент времени t.

{

return top + 1;

}

};

#endif

**Модуль “postfix.h”**

#ifndef \_\_POSTFIX\_H\_\_

#define \_\_POSTFIX\_H\_\_

#include <iostream>

#include <string>

#include <vector>

#include "stack.h"

using namespace std;

class TPostfix

{

private:

string infix;

string postfix;

double \*value; //для хранения значений переменных выражения

TStack<char> stackOperation; // для преобразования в постфикс и проверки на корректность

TStack<string> stackArguments; // для хранение аргументов

TStack<double> stackValue; //для хранения соответствующих значений

TStack<double> stackValuePostfix; //для высчитывания результата

public:

TPostfix(string \_infix);

string GetInfix();

string GetPostfix();

void ChangeInfix(string \_infix);

bool IsCorrect(); //проверка на корректность выражения

void ToPostfix(); // получение постфикса

void ReadArguments(); //ввод значений переменных

double Calculate(); // вычисление

};

#endif

**Модуль “postfix.cpp”**

#include "postfix.h"

#include "stack.h"

using namespace std;

bool IsOperation(char elem);

bool IsOperationForCorrect(char elem);

int Priority(char elem);

double Result(double op1, double op2, char operat);

bool IsNumber(char elem);

TPostfix::TPostfix(string \_infix) : infix(\_infix), postfix(""), value(nullptr) {}

string TPostfix::GetInfix()

{

return infix;

}

string TPostfix::GetPostfix()

{

return postfix;

}

void TPostfix::ChangeInfix(string \_infix)

{

infix = \_infix;

postfix.clear();

}

bool TPostfix::IsCorrect() //проверка на корректность выражения

{

int flag = 0;

stackOperation.Clear();

if (IsOperationForCorrect(infix[0]) || IsOperationForCorrect(infix[infix.size() - 1]))//выражение не начинается и не заканчивается операцией

return false;

for (int i = 1; i < infix.size() - 1; i++) // в выражении нет двух операций подряд

if (IsOperationForCorrect(infix[i]) && IsOperationForCorrect(infix[i + 1]))

return false;

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

{

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

stackOperation.Push(1);

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

{

if (stackOperation.IsEmpty())

flag = 1;

stackOperation.Erase();

}

}

if (flag)

return false;

return (stackOperation.IsEmpty()) ? true : false;

}

void TPostfix::ToPostfix() // получение постфикса

{

stackOperation.Clear();

int a = 0;

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

{

if (!IsOperation(infix[i]))

{

postfix += infix[i];

if (i == infix.size() - 1 || IsOperation(infix[i + 1]))

postfix += ' ';

}

else

{

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

{

if (Priority(infix[i]) == 0 || stackOperation.IsEmpty() || Priority(infix[i]) > Priority(stackOperation.PopWithoutDelete()))

stackOperation.Push(infix[i]);

else

{

a = stackOperation.Length();

for (int j = 0; j < a; j++)

{

if (Priority(stackOperation.PopWithoutDelete()) >= Priority(infix[i]))

{

postfix += stackOperation.Pop();

postfix += ' ';

}

else

break;

}

stackOperation.Push(infix[i]);

}

}

else

{

a = stackOperation.Length();

for (int j = 0; j < a; j++)

{

if (stackOperation.PopWithoutDelete() != '(')

{

postfix += stackOperation.Pop();

postfix += ' ';

}

else

{

stackOperation.Erase();

break;

}

}

}

}

}

a = stackOperation.Length();

for (int j = 0; j < a; j++)

{

postfix += stackOperation.Pop();

if (j != a - 1)

postfix += ' ';

}

}

void TPostfix::ReadArguments() //ввод значений переменных

{

stackArguments.Clear();

stackValue.Clear();

string ss;

for (int i = infix.size() - 1; i >= 0; i--)

if (!IsOperation(infix[i]) && !IsNumber(infix[i]) && infix[i] != '.')

{

ss += infix[i];

if (i != 0 && !IsOperation(infix[i - 1]) && !IsNumber(infix[i - 1]))

continue;

reverse(ss.begin(), ss.end());

stackArguments.Push(ss);

ss.clear();

}

int res = stackArguments.Length();

value = new double[res];

if (res)

cout << "Введите значения переменных." << endl;

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

{

cout << stackArguments.Pop() << "=";

cin >> value[i];

}

for (int i = res - 1; i >= 0; i--)

stackValue.Push(value[i]);

}

double TPostfix::Calculate() // вычисление

{

stackValuePostfix.Clear();

double tmp = 0;

string ss;

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

{

if (postfix[i] != ' ')

{

if (!IsOperation(postfix[i]) && !IsNumber(postfix[i]) && postfix[i] != '.')

{

if (i != postfix.size() - 1 && postfix[i + 1] != ' ')

continue;

stackValuePostfix.Push(stackValue.Pop());

}

else if (IsNumber(postfix[i]) || postfix[i] == '.')

{

ss += postfix[i];

if (i != postfix.size() - 1 && IsNumber(postfix[i + 1]))

continue;

if (postfix[i + 1] == '.')

continue;

tmp = stod(ss);

stackValuePostfix.Push(tmp);

ss.clear();

}

else

{

tmp = Result(stackValuePostfix.Pop(), stackValuePostfix.Pop(), postfix[i]);

stackValuePostfix.Push(tmp);

}

}

else continue;

}

return stackValuePostfix.Pop();

}

bool IsOperation(char elem)

{

return (elem == '(' || elem == ')' || elem == '+' || elem == '-' || elem == '\*' || elem == '/') ? true : false;

}

bool IsOperationForCorrect(char elem)

{

return (elem == '+' || elem == '-' || elem == '\*' || elem == '/') ? true : false;

}

int Priority(char elem)

{

if (!IsOperation(elem))

throw "data is not correct";

if (elem == '(')

return 0;

else if (elem == ')')

return 1;

else if (elem == '+' || elem == '-')

return 2;

return 3;

}

double Result(double op1, double op2, char operat)

{

if (operat == '+')

return op1 + op2;

else if (operat == '-')

return op1 - op2;

else if (operat == '\*')

return op1 \* op2;

else

return op1 / op2;

}

bool IsNumber(char elem)

{

for (char i = '0'; i <= '9'; i++)

if (elem == i)

return true;

return false;

}

**Модуль “sample\_postfix.cpp”**

#include <string>

#include "postfix.h"

using namespace std;

int main()

{

int a = 0;

setlocale(LC\_CTYPE, "Russian");

string expression;

m:cout << "Введите арифметическое выражение(без пробелов)" << endl;

cout << "Для выделения дробной части используйте '.' Например: 5.2" << endl;

cout << "Разрешается использование переменных, название произвольное и не ограничено в длине. Пример: res+sum" << endl;

cin >> expression;

TPostfix postfix(expression);

n:if (!postfix.IsCorrect())

{

cout << "Выражение не корректно, повторите." << endl;

goto m;

}

postfix.ToPostfix();

k:postfix.ReadArguments();

cout << "Результат вычисления: " << postfix.Calculate() << endl;

cout << "Постфиксная форма: " << postfix.GetPostfix() << endl;

cout << "Выберите дальнейшнее действие:" << endl;

cout << "1. Ввести другое арифметическое выражение" << endl;

cout << "2. Ввести другие значения переменных(если они присутствуют)" << endl;

cout << "3. Выйти из программы" << endl;

cin >> a;

switch (a)

{

case 1:

{

cin >> expression;

postfix.ChangeInfix(expression);

goto n;

break;

}

case 2:

{

goto k;

break;

}

case 3: break;

}

return 0;

}

**Google Test**

**Модуль “test\_tpostfix.cpp”**

#include "postfix.h"

#include "postfix.cpp"

#include <gtest.h>

TEST(TPostfix, can\_create\_postfix)

{

ASSERT\_NO\_THROW(TPostfix p("0"));

}

TEST(TPostfix, can\_change\_infix)

{

TPostfix p("a+b\*c");

p.ChangeInfix("a+0.5-1");

EXPECT\_EQ("a+0.5-1", p.GetInfix());

}

TEST(TPostfix, can\_not\_start\_infix\_the\_operation)

{

TPostfix p("+a+1");

EXPECT\_EQ(false, p.IsCorrect());

}

TEST(TPostfix, can\_not\_finish\_infix\_the\_operation)

{

TPostfix p("a+1+");

EXPECT\_EQ(false, p.IsCorrect());

}

TEST(TPostfix, infix\_has\_not\_two\_operation\_in\_a\_row)

{

TPostfix p("a+\*1");

EXPECT\_EQ(false, p.IsCorrect());

}

TEST(TPostfix, parentheses\_is\_not\_correct)

{

TPostfix p("(a+1))");

EXPECT\_EQ(false, p.IsCorrect());

}

TEST(TPostfix, parentheses\_is\_correct)

{

TPostfix p("(a+1)\*(b+1)");

EXPECT\_EQ(true, p.IsCorrect());

}

TEST(TPostfix, expression\_is\_correct\_when\_change\_infix)

{

TPostfix p("(a+1)\*(b+1)");

p.ChangeInfix("(a+b)/(e-f)");

EXPECT\_EQ(true, p.IsCorrect());

}

TEST(TPostfix, postfix\_change\_when\_change\_infix)

{

TPostfix p("(a+1)\*(b+1)");

p.ChangeInfix("(a+b)/(e-f)");

p.ToPostfix();

EXPECT\_EQ("a b + e f - /", p.GetPostfix());

}

TEST(TPostfix, transformation\_infix\_in\_postfix\_is\_true\_only\_numbers)

{

TPostfix p("(5+1)\*(7+1)");

p.ToPostfix();

EXPECT\_EQ("5 1 + 7 1 + \*", p.GetPostfix());

}

TEST(TPostfix, transformation\_infix\_in\_postfix\_is\_true\_only\_variables)

{

TPostfix p("(a+b\*c)-(k\*d-p)");

p.ToPostfix();

EXPECT\_EQ("a b c \* + k d \* p - -", p.GetPostfix());

}

TEST(TPostfix, transformation\_infix\_in\_postfix\_is\_true\_variables\_and\_numbers\_mixed)

{

TPostfix p("(5+4\*v)/5.0");

p.ToPostfix();

EXPECT\_EQ("5 4 v \* + 5.0 /", p.GetPostfix());

}

TEST(TPostfix, can\_use\_only\_long\_variables)

{

TPostfix p("(alpha+1)\*(betta+1)");

p.ToPostfix();

EXPECT\_EQ("alpha 1 + betta 1 + \*", p.GetPostfix());

}

TEST(TPostfix, can\_use\_variables\_short\_and\_long)

{

TPostfix p("(num\*c+5.0)/denominator");

p.ToPostfix();

EXPECT\_EQ("num c \* 5.0 + denominator /", p.GetPostfix());

}

TEST(TPostfix,can\_calculate\_expression)

{

TPostfix p("(5+1)\*(10+1)");

p.ToPostfix();

p.ReadArguments();

EXPECT\_EQ(66, p.Calculate());

}

**Модуль “test\_tstack.cpp”**

#include "stack.h"

#include <gtest.h>

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

{

ASSERT\_NO\_THROW(TStack<int> st(5));

}

TEST(TStack, can\_not\_create\_stack\_with\_negative\_size)

{

ASSERT\_ANY\_THROW(TStack<int> st(-1));

}

TEST(TStack, can\_not\_create\_stack\_with\_large\_size)

{

ASSERT\_ANY\_THROW(TStack<int> st(MaxStackSize + 1));

}

TEST(TStack, two\_different\_stacks\_have\_different\_memories)

{

TStack<int> s1(2), s2(3);

EXPECT\_NE(&s1, &s2);

}

TEST(TStack, check\_stack\_for\_empty)

{

TStack<int> s1(5);

EXPECT\_EQ(true, s1.IsEmpty());

}

TEST(TStack, check\_stack\_for\_full)

{

TStack<int> s1(5);

for (int i = 0; i < 5; i++)

s1.Push(i);

EXPECT\_EQ(true, s1.IsFull());

}

TEST(TStack, can\_clear\_stack)

{

TStack<int> s1(5);

for (int i = 0; i < 5; i++)

s1.Push(i);

s1.Clear();

EXPECT\_EQ(true, s1.IsEmpty());

}

TEST(TStack, can\_erase\_last\_element\_in\_stack)

{

TStack<int> s1(5);

for (int i = 0; i < 5; i++)

s1.Push(i);

s1.Erase();

EXPECT\_EQ(false, s1.IsFull());

}

TEST(TStack, can\_take\_element\_in\_stack\_pop\_with\_corrected\_value)

{

TStack<int> s1(5);

for (int i = 0; i < 5; i++)

s1.Push(i);

EXPECT\_EQ(4, s1.Pop());

}

TEST(TStack, element\_is\_delete\_on\_pop)

{

TStack<int> s1(5);

for (int i = 0; i < 5; i++)

s1.Push(i);

int a = s1.Pop();

EXPECT\_EQ(false, s1.IsFull());

}

TEST(TStack, can\_view\_last\_element\_in\_stack\_without\_delete\_him)

{

TStack<int> s1(5);

for (int i = 0; i < 5; i++)

s1.Push(i);

int a = s1.PopWithoutDelete();

EXPECT\_EQ(true, s1.IsFull());

}

TEST(TStack,method\_pop\_without\_delete\_return\_correct\_value)

{

TStack<int> s1(5);

for (int i = 0; i < 5; i++)

s1.Push(i);

EXPECT\_EQ(4, s1.PopWithoutDelete());

}

TEST(TStack,can\_know\_length\_now)

{

TStack<int>s1(5);

s1.Push(1);

EXPECT\_EQ(1, s1.Length());

}