Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский 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 3821Б1ФИ3

Шушпанов Н.М.

**Проверил**: заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И.Г.

Нижний Новгород

2022

Содержание

[Введение 3](#_Toc86935064)

[1. Постановка задачи 4](#_Toc86935065)

[2. Руководство пользователя 5](#_Toc86935066)

[3.1. Описание структуры программы 6](#_Toc86935067)

[3.2. Описание алгоритмов 6](#_Toc86935068)

[3.2.1 Алгоритм перевода выражения в постфиксную запись: 6](#_Toc86935069)

[3.2.2Алгоритм подсчета постфиксной записи: 7](#_Toc86935070)

[4. Результаты экспериментов 8](#_Toc86935071)

[Заключение 9](#_Toc86935072)

[Литература 10](#_Toc86935073)

[Приложение 11](#_Toc86935074)

# Введение

В современном мире множество процессов жизни человека связано с вычислением и обработкой информации, в частности, с работой с арифметическими выражениями, как с простыми, так и более сложными. Если с вычислением более простых выражений человек не плохо справляется, то с более сложными могут уже возникать ошибки из-за невнимательности, усталости и т.д.

Поэтому в данной лабораторной работе будет рассматриваться возможность вычисления выражений с помощью компьютера, который может справиться с этим быстрее и эффективнее. Для этого также будет разработана вспомогательная структура данных «cтек», с помощью которой и будет реализован алгоритм перевода инфиксной формы арифметического выражения в постфиксную, а также подсчёт.

# Постановка задачи

Разработать программу, которая будет производить вычисления введенного арифметического выражения с помощью использования вспомогательной структуры данных.

Входные данные:

* Арифметическое выражение в инфиксной форме, которое может содержать такие операторы, как “+”, “-”, “\”, “\*”, а также “(”,“)”.

Выходные данные:

* Арифметическое выражение в постфиксной форме («Обратная польская запись»).
* Вычисленное арифметическое выражение.

То есть при выполнении работы мы решим следующие задачи:

* Разработка интерфейса шаблонного класса **TStack**.
* Реализация методов шаблонного класса **TStack**.
* Разработка интерфейса класса **TPostfix** для работы с постфиксной формой.
* Реализация методов класса **TPostfix**.
* Разработка и реализация тестов для классов **TStack** и **TPostfix** на базе Google Test.

# Руководство пользователя

Программа имеет вид консольного приложения.

Сначала программа предложит ввести пользователю арифметическое выражение в инфиксной форме (рис. 1).

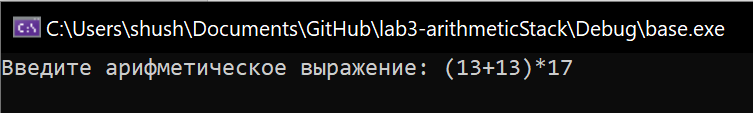


Рис. 1. Ввод арифметического выражения

Далее программа выведет выражение в постфиксной форме (рис. 2).



Рис. 2. Вывод выражения в постфиксной форме

В итоге будет выведено вычисленное арифметическое выражение (рис. 3).



Рис. 3. Вывод результата

1. **Руководство программиста**

## Описание структуры программы

Программа реализована на языке программирования C++ и состоит из:

* gtest — библиотека Google Test.
* base — каталог с основным проектом лабораторной работы.
* base\_test — каталог с проектом с модульными тестами.
* README.md — информация о проекте.
* lab3-arithmeticStack.sln - файл с решением (solution) для Microsoft Visual Studio 2010.
* служебные файлы
* gitignore — перечень расширений файлов, игнорируемых Git при добавлении файлов в репозиторий.
* stack.h – интерфейс и реализация методов для работы c классом TStack.
* postfix.h – заголовочный файл для работы с классом TPostfix.
* postfix.cpp – реализация методов класса TPostfix.
* sample\_postfix.cpp.

**Класс TStack:**

* T\* pMem – указатель на память, выделенную для хранения стека.
* size – хранит размер выделенной памяти (тип int).
* top – хранит индекс элемента массива, в котором хранится последнее добавленное значение стека (тип int).
* DataCount – хранит количество запомненных в стеке значений (тип int).
* Класс TStack:
* TStack(int \_size) – конструктор класса
* ~TStack(int \_size) – деструктор класса
* GetNextIndex – возвращает индекс следующего элемента в стеке
* IsEmpty – проверка на пустоту стека
* IsFull – проверка на полноту стека
* Put – добавление элемента в стек

**Класс TPostfix:**

* infix – хранит строку в виде инфиксной формы
* postfix – хранит строку в виде постфиксной формы
* Priority – возвращает приоритет оператора
* GetInfix – возвращает строку в инфиксной форме
* GetPostfix – возвращает строку в постфиксной форме
* ToPostfix – переводит из инфиксной формы в постфиксную
* Calculate – вычисление по постфиксной форме

## Описание алгоритмов

3.2.1 Алгоритм перевода выражения в постфиксную запись:

1.Константы и переменные кладутся в формируемую запись в порядке их появления в исходном массиве. При появлении операции в исходном массиве:

* если в стеке нет операций или верхним элементом стека является открывающая скобка, операции кладётся в стек;
* если новая операции имеет больший приоритет, чем верхняя операции в стеке, то новая операции кладётся в стек;
* если новая операция имеет меньший или равный приоритет, чем верхняя операции в стеке, то операции, находящиеся в стеке, до ближайшей открывающей скобки или до операции с приоритетом меньшим, чем у новой операции, перекладываются в формируемую запись, а новая операции кладётся в стек.

2. Открывающая скобка кладётся в стек.

3. Закрывающая скобка выталкивает из стека в формируемую запись все операции до ближайшей открывающей скобки, открывающая скобка удаляется из стека.

4. После того, как мы добрались до конца исходного выражения, операции, оставшиеся в стеке, перекладываются в формируемое выражение.

3.2.2 Алгоритм подсчета постфиксной записи:

1. Обработка входного символа:

* если на вход подан операнд, он помещается на вершину стека;
* если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления.

Результат выполненной операции кладётся на вершину стека.

2. Если входной набор символов обработан не полностью, перейти к шагу 1.

3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека.

# Результаты экспериментов

Для выявления ошибок в работе программы были реализованы тесты на основе фреймворка для написания автоматических тестов Google Test.

Примеры тестов, проверяющие класс TStack (с полным списком можно ознакомиться в приложении):

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

{

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

}

TEST(TStack, cant\_create\_stack\_with\_negative\_length)

{

ASSERT\_ANY\_THROW(TStack<int> v(-5));

}

TEST(TStack, cant\_create\_too\_large\_stack)

{

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

}

TEST(TStack, cant\_create\_stack\_with\_zero\_length)

{

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

}

TEST(TStack, cant\_put\_to\_FULL\_stack)

{

TStack<int> v(2);

v.Put(1);

v.Put(2);

ASSERT\_ANY\_THROW(v.Put(4));

}

Примеры тестов, проверяющие класс TPostfix (с полным списком можно ознакомиться в приложении):

TEST(TPostfix, can\_create\_postfix)

{

string expression = "2+2";

ASSERT\_NO\_THROW(TPostfix p(expression));

}

TEST(TPostfix, simple\_multiplication\_test\_num1)

{

string expression = "5\*5";

int result = 25;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

TEST(TPostfix, simple\_multiplication\_test\_num2\_with\_scobe)

{

string expression = "(5\*5)\*6";

int result = 150;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

# Заключение

Для поставленной задачи была разработана программа, реализующая вычисление арифметических операций с помощью специальной структуры хранения данных – стека и перевода инфиксной формы выражения в постфиксную.

# Литература

1. Сайт ru.wikipedia.org Обратная польская запись - https://ru.wikipedia.org/wiki/Обратная\_польская\_запись
2. Сайт habr.com. Обратная польская запись - <https://habr.com/ru/post/100869/>

# Приложение

Код исходной программы:

sample\_postfix.cpp

#include <iostream>

#include <string>

#include "postfix.h"

using namespace std;

int main()

{

string expression;

double res;

setlocale(LC\_ALL, "Russian");

cout << "Введите арифметическое выражение: ";

cin >> expression;

cout << expression << endl;

TPostfix postfix(expression);

cout << "Арифметическое выражение: " << postfix.GetInfix() << endl;

postfix.ToPostfix();

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

cout << postfix.Calculate() << endl;

return 0;

}

stack.h

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

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

#include <cstddef>

const int MaxStackSize = 100;

template <class T>

class TStack

{

T\* pMem;

int size;

int top;

int DataCount;

public:

TStack(int \_size) {

size = \_size;

top = 0;

if ((size < 1) || (size > MaxStackSize)) { throw size; }

DataCount = 0;

pMem = new T[size];

}

~TStack(){ delete[] pMem;}

TStack& operator=(const TStack& tstack);

int GetNextIndex(int index);

int IsEmpty(void) const;

int IsFull(void) const;

void Put(const T& Val);

virtual T Get ( void ) ;

T Pop();

};

template<class T>

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

if (this == &tstack) { return \*this;}

pMem = tstack.pMem;

return \*this;

}

template<class T>

inline int TStack<T>::GetNextIndex(int index) { return ++index; }

template<class T>

inline int TStack<T>::IsEmpty(void) const { return(DataCount == 0);}

template<class T>

inline int TStack<T>::IsFull(void) const { return (DataCount == size); }

template<class T>

inline void TStack<T>::Put(const T& Val){

if (IsFull()) { throw (DataCount); }

pMem[top] = Val;

DataCount++;

top++;

}

template<class T>

inline T TStack<T>::Pop() {

if (this->IsEmpty()) { throw exception(); }

top--;

DataCount--;

T tmp = pMem[top];

pMem[top] = NULL;

return tmp;

}

template<class T>

inline T TStack<T>::Get(void) {

if (IsEmpty()) { throw (top); }

return pMem[top - 1];

}

#endif

postfix.h

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

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

#include <string>

#include "stack.h"

using namespace std;

class TPostfix

{

string infix;

string postfix;

public:

TPostfix(string Value) { infix = Value; }

TPostfix() { infix = "a + b"; }

int Priority(char val);

size\_t checkStringIsANumber(string inString);

char getEndOfString(); // return '\0'

string GetInfix() { return infix; }

string GetPostfix() { return postfix; }

string ToPostfix();

double Calculate();

};

#endif

postfix.cpp

#include "postfix.h"

#include "stack.h"

#include <iostream>

#include <cstdlib>

using namespace std;

int TPostfix::Priority(char val){

if ((val == '\*') || (val == '/')) { return 2; }

else if ((val == '+') || (val == '-')) { return 1; }

return 0;

}

size\_t TPostfix::checkStringIsANumber(string inString) {

unsigned int flag = inString.find\_first\_not\_of("0123456789");

return flag;

}

char TPostfix::getEndOfString() { return '\0'; }

string TPostfix::ToPostfix(){

char result[MaxStackSize];

unsigned int resultPosition = 0;

TStack<char> stack\_(MaxStackSize);

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

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

(infix[i] != '-') && (infix[i] != '\*') && (infix[i] != '/')){

result[resultPosition] = infix[i];

resultPosition++;

result[resultPosition] = getEndOfString();

}

if (infix[i] == '(') { stack\_.Put(infix[i]); }

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

while (stack\_.Get() != '('){

result[resultPosition] = ' ';

resultPosition++;

result[resultPosition] = stack\_.Pop();

resultPosition++;

result[resultPosition] = getEndOfString();

}

stack\_.Pop();

}

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

result[resultPosition] = ' ';

resultPosition++;

if (stack\_.IsEmpty() || stack\_.Get() == '(') { stack\_.Put(infix[i]); }

else if (Priority(infix[i]) > Priority(stack\_.Get())) { stack\_.Put(infix[i]); }

else{

while (!stack\_.IsEmpty() && Priority(infix[i]) <= Priority(stack\_.Get())){

result[resultPosition] = stack\_.Pop();

resultPosition++;

result[resultPosition] = getEndOfString();

}

result[resultPosition] = ' ';

resultPosition++;

stack\_.Put(infix[i]);

}

}

}

while (stack\_.IsEmpty() == 0){

result[resultPosition] = ' ';

resultPosition++;

result[resultPosition] = stack\_.Pop();

resultPosition++;

result[resultPosition] = getEndOfString();

}

postfix = result;

return postfix;

};

double TPostfix::Calculate(){

double storageOfOperands[MaxStackSize];

string intermediateResult;

double variableValues;

int j = 0;

TStack<double> stack(MaxStackSize);

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

if ((postfix[i] != '+') && (postfix[i] != '-') && (postfix[i] != '\*') && (postfix[i] != '/')){

if (postfix[i] != ' ') { intermediateResult = intermediateResult + postfix[i]; }

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

(postfix[i-1] != '-') && (postfix[i-1] != '\*') && (postfix[i-1] != '/')){

size\_t flag = checkStringIsANumber(intermediateResult);

if ((flag == std::string::npos) && (postfix[i] == ' ')) {

storageOfOperands[j] = stoi(intermediateResult);

}

else {

cout << postfix[i - 1] << "=";

cin >> variableValues;

storageOfOperands[j] = variableValues;

}

stack.Put((double)storageOfOperands[j]);

intermediateResult = "";

j++;

}

}

else {

double tmp1 = stack.Pop();

double tmp2 = stack.Pop();

if (postfix[i] == '+') { stack.Put(tmp2 + tmp1); }

if (postfix[i] == '-') { stack.Put(tmp2 - tmp1); }

if (postfix[i] == '/') { stack.Put(tmp2 / tmp1); }

if (postfix[i] == '\*') { stack.Put(tmp2 \* tmp1); }

}

}

return stack.Pop();

};

test\_tpostfix.cpp

#include "postfix.h"

#include "postfix.cpp"

#include <gtest.h>

TEST(TPostfix, can\_create\_postfix)

{

string expression = "2+2";

ASSERT\_NO\_THROW(TPostfix p(expression));

}

TEST(TPostfix, simple\_multiplication\_test\_num1)

{

string expression = "5\*5";

int result = 25;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

TEST(TPostfix, simple\_multiplication\_test\_num2\_with\_scobe)

{

string expression = "(5\*5)\*6";

int result = 150;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

TEST(TPostfix, simple\_addition\_test\_num1)

{

string expression = "5+5";

int result = 10;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

TEST(TPostfix, simple\_addition\_test\_num2\_with\_scobe)

{

string expression = "(5+5)+5";

int result = 15;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

TEST(TPostfix, simple\_divison\_test\_num1)

{

string expression = "5/5";

int result = 1;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

TEST(TPostfix, simple\_divison\_test\_num1\_with\_scobe)

{

string expression = "(20/2)/2";

int result = 5;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

TEST(TPostfix, simple\_subtraction\_test\_num1)

{

string expression = "5-5";

int result = 0;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

TEST(TPostfix, simple\_subtraction\_test\_num2\_with\_scobe)

{

string expression = "(5-5)-5";

int result = -5;

TPostfix postfix(expression);

postfix.ToPostfix();

EXPECT\_EQ(result, postfix.Calculate());

}

test\_tstack.cpp

#include "stack.h"

#include <gtest.h>

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

{

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

}

TEST(TStack, cant\_create\_stack\_with\_negative\_length)

{

ASSERT\_ANY\_THROW(TStack<int> v(-5));

}

TEST(TStack, cant\_create\_too\_large\_stack)

{

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

}

TEST(TStack, cant\_create\_stack\_with\_zero\_length)

{

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

}

TEST(TStack, cant\_put\_to\_FULL\_stack)

{

TStack<int> v(2);

v.Put(1);

v.Put(2);

ASSERT\_ANY\_THROW(v.Put(4));

}

TEST(TStack, cant\_get\_of\_CLEAR\_stack)

{

TStack<int> v(5);

ASSERT\_ANY\_THROW(v.Get());

}

TEST(TStack, can\_get\_of\_stack\_about\_one)

{

TStack<int> v(5);

v.Put(4);

EXPECT\_EQ(4, v.Get());

}

TEST(TStack, can\_get\_of\_stack\_about\_two)

{

TStack<int> v(5);

v.Put(4);

v.Put(7);

EXPECT\_EQ(7, v.Get());

}

TEST(TStack, can\_get\_of\_stack\_about\_three)

{

TStack<int> v(5);

v.Put(4);

v.Put(1);

v.Put(10);

EXPECT\_EQ(10, v.Get());

}

TEST(TStack, IsEmpty\_false\_of\_full\_stack)

{

TStack<int> v(3);

v.Put(5);

v.Put(5);

ASSERT\_FALSE(v.IsEmpty());

}

TEST(TStack, IsEmpty\_true\_of\_clear\_stack)

{

TStack<int> v(2);

ASSERT\_TRUE(v.IsEmpty());

}

TEST(TStack, IsEmpty\_false\_of\_any\_stack)

{

TStack<int> v(2);

v.Put(4);

ASSERT\_FALSE(v.IsEmpty());

}

TEST(TStack, IsFull\_true\_of\_full\_stack)

{

TStack<int> v(2);

v.Put(4);

v.Put(4);

ASSERT\_TRUE(v.IsFull());

}

TEST(TStack, IsFull\_false\_of\_clear\_stack)

{

TStack<int> v(2);

ASSERT\_FALSE(v.IsFull());

}

TEST(TStack, IsFull\_false\_of\_any\_stack)

{

TStack<int> v(2);

v.Put(4);

ASSERT\_FALSE(v.IsFull());

}