Министерство образования и науки 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 1703-2

\_\_\_\_\_\_\_\_\_\_\_\_ Хайрутдинов Э.Ш

Подпись

**Проверил**: доцент кафедры МОСТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сысоев А.В.

Подпись

Нижний Новгород

2018

**Содержание**

[**Введение** 3](#_Toc529998982)

[**Постановка задачи** 4](#_Toc529998983)

[**Руководство пользователя** 5](#_Toc529998984)

[**Руководство программиста** 7](#_Toc529998985)

[**Эксперимент** 8](#_Toc529998986)

[**Заключение** 9](#_Toc529998987)

[**Литература** 10](#_Toc529998988)

[**Приложение** 11](#_Toc529998989)

# **Введение**

Целью данной работы является совершенствование навыков разработки и использования классов и их систем в языке программирования С++ путем реализации динамической структуры данных Стек. Также целью работы является освоение алгоритма вычисления арифметического выражения за один просмотр. Для реализации данного алгоритма необходима чтобы запись была в постфиксной форме.

# **Постановка задачи**

Передо мной была поставлена задача написать класс, которая реализует классы Стек и Постфикс и их функции:

* Методы добавления и удаления элемента из стека.
* Методы проверки стека на полноту и пустоту.
* Метод перевода арифметического выражения в постфикс и его проверки на корректность.
* Метод подсчета арифметического выражения в постфиксном виде.

# **Руководство пользователя**

1. Перед Вами появилась консоль с меню(рис.1).

Рис.1 «Стартовый вид консоли»

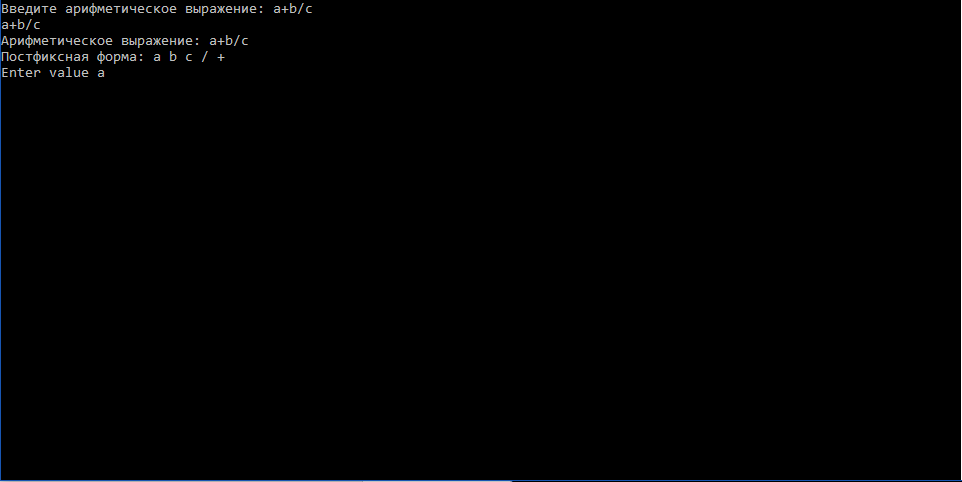
1. Введите выражение. Можете ввести как формулу, так и численное выражение.
   1. Если Вы ввели формулу, программа переведет формулу в постфиксную форму и попросит ввести значения операндов(рис.2).

Рис.2 «Ввод выражения в буквенном виде»

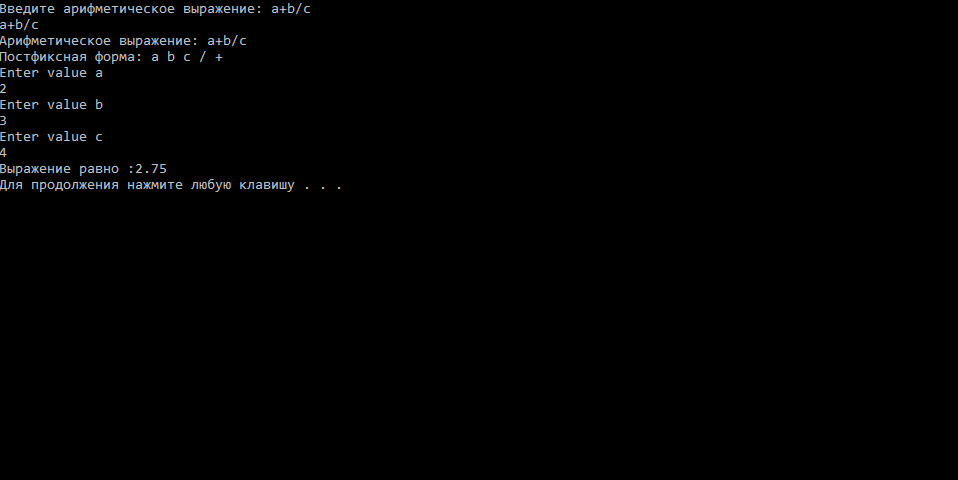
* 1. Когда Вы введете значение, то программа вычислит значение выражения(рис.3).

Рис.3 «Ввод значений операндов и вычисление выражения»

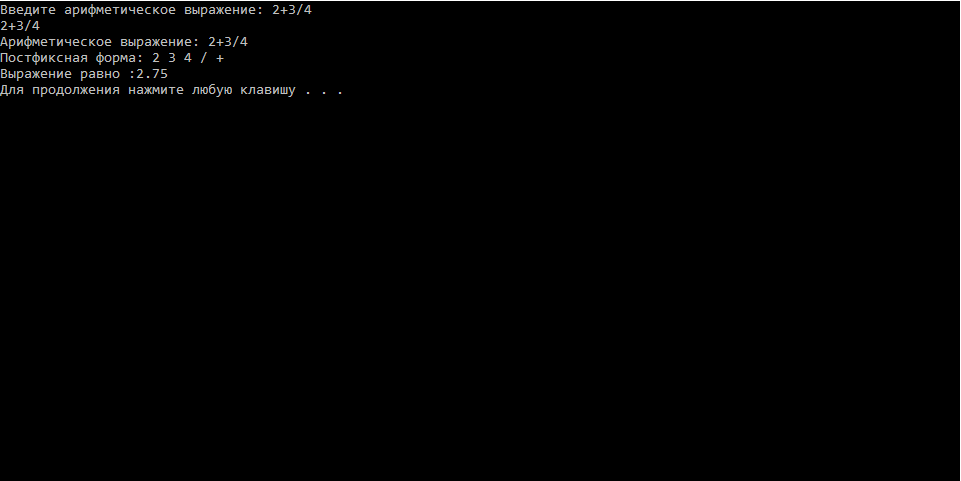
1. Если Вы ввели численное выражение, то программа сразу переведет в постфикс и вычислит значение выражение(рис.4).

Рис.4 «Ввод численного выражения.

1. Чтобы продолжить нажмите на любую кнопку (кроме кнопки выключения).

# **Руководство программиста**

Класс TStack имеет методы, которые осуществляют проверку стека на полноту и пустоту, методы добавления и удаление элемента из стека.

**Bool IsEmpty() –** проверка на пустоту.

**Bool IsFull() –** проверка на полноту.

**Void Push(constTelem&val) –** метод,добавляющий элемент в стек.

**Telem Pop() –** метод, удаляющий верхний элемент из стека.

**Telem Back()-** метод, возвращающий верхний элемент.

Класс TPostfixимеет методы, которые проверяют корректность введенных данных, производят перевод обычной записи в постфиксную и вычисления выражения, а также вспомогательные методы.

**Boo lCorrectRecord() –** метод, проверяющий корректность вводимого выражения.

**Void ToPostfix() –** метод, переводящий выражение в постфиксную форму.

**Void Converter(string&str) –** метод, переводящий буквенное выражение в численное.

**Double Calculate() –** метод, вычисляющий выражение в постфиксной форме.

Вспомогательные методы:

**Void SetInfix(string \_infix) –** метод, принимает на вход строку с выражением и присваивает это значение полю инфикс.

**String GetInfix() –** метод, возвращающий выражение в инфиксной форме.

**boolIs Operations(chars) –** метод, проверяющий символ на принадлежность арифметическим операциям.

**boolIs Operand(chars) –** метод, проверяющий символ на принадлежность буквенным операндам.

**String GetPostfix() –** метод, возвращающий выражение в постфиксной форме.

**Int Compare(char \_t1) –** метод, установления приоритет операциям.

# **Эксперимент**

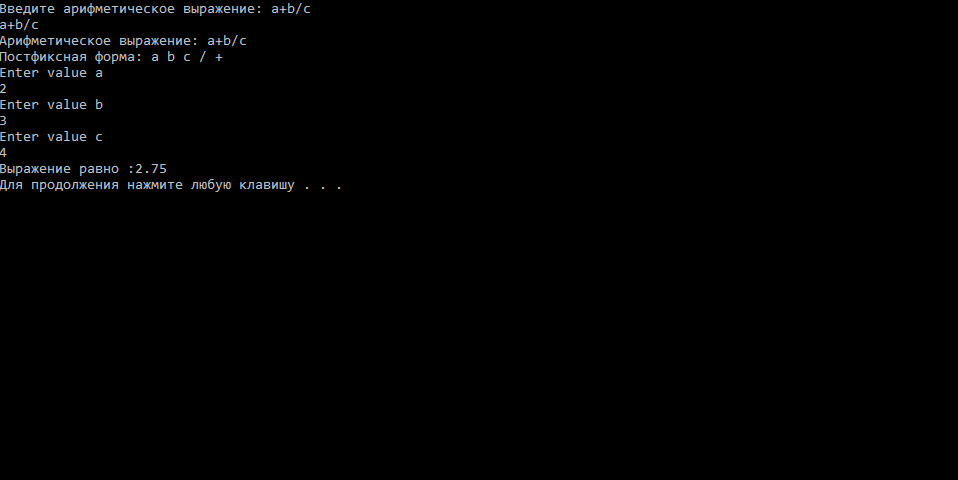
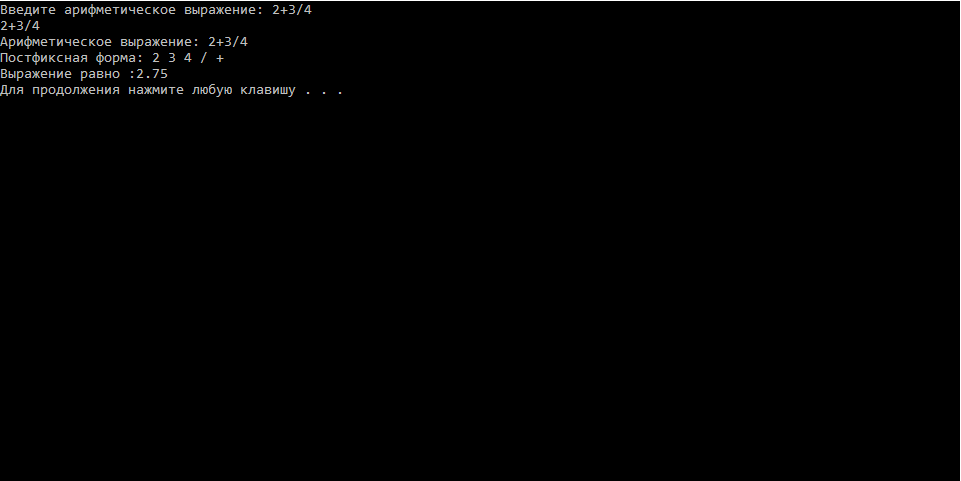
В результате эксперимента, показанного на рис.5 и рис 6, мы видим, что программа считает правильно выражение, введенное как в численном, так и в буквенном виде. Программа функционирует и работает правильно.

Рис.5 «Ввод формулы, значений операндов и вычисление выражения»

Рис.6 «Ввод численного выражения»

# **Заключение**

В результате работы было написано 2 класса и программа, использующая эти классы.

В ходе работы я изучил новый абстрактный тип данных Stack, изучил алгоритмы преобразования выражения в постфикс.

# **Литература**

1. http://cpp.mazurok.com/tag/стек/
2. http://www.lib.unn.ru/students/src/Pract\_ADS.pdf
3. http://cpp.mazurok.com/tag/постфиксная-форма/
4. Бьерн Страуструп Язык программирования С++
5. Герберт Шилдт С++ базовый курс.
6. Т. Пратт, М. Зелковиц. Языки программирования: разработка и реализация = Terrence W. Pratt, Marvin V. Zelkowitz. Programming Languages: Design and Implementation. — 4-еиздание. — Питер, 2002. — 688 с. — (Классика Computer Science). — 4000 экз. — ISBN 5-318-00189-0.

# **Приложение**

#include "postfix.h"

#include "stack.h"

#include "iostream"

#include "vector"

#include <cctype>

void TPostfix::ToPostfix()

{

if (CorrectRecord() == false)

throw "error";

TStack<char> op(infix.size());

postfix = "";

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

{

if (isalpha(infix[i]) || isdigit(infix[i]))

{

postfix += infix[i];

while (isalpha(infix[i + 1]) || isdigit(infix[i + 1]))

{

postfix += infix[i + 1];

i++;

}

postfix += ' ';

}

else

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

op.Push(infix[i]);

else

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

{

while (op.Back() != '(')

{

postfix += op.Pop();

postfix += ' ';

}

op.Pop();

}

else

if (op.IsEmpty() == true)

{

op.Push(infix[i]);

}

else

if (Compare(infix[i]) <= Compare(op.Back()))

{

postfix += op.Pop();

if (Compare(infix[i]) <= Compare(op.Back()))

{

postfix += op.Pop();

postfix += ' ';

}

postfix += ' ';

op.Push(infix[i]);

}

else

op.Push(infix[i]);

}

while (op.IsEmpty() == false)

{

postfix += op.Pop();

postfix += ' ';

}

}

void TPostfix::SetInfix(string \_infix)

{

infix = \_infix;

}

void TPostfix::Converter(string &str)

{

string tmp;

for (inti = 0; i<str.size(); i++)

{

if (IsOperand(str[i]))

{

cout<< "Enter value " <<str[i] <<endl;

cin>>tmp;

char sym = str[i];

int j = i;

while (j <str.size())

{

if (sym == str[j])

{

str.erase(j, 1);

str.insert(j, tmp);

j += tmp.size();

}

else j++;

}

}

}

}

double TPostfix::Calculate()

{

Converter(postfix);

TStack<double> value(postfix.size());

string tmp;

double tmp1;

double tmp2;

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

{

if (postfix[i] == ' ')

continue;

if (IsOperations(postfix[i]) == false)

{

while (postfix[i] != ' ')

{

tmp += postfix[i];

i++;

}

value.Push(atof(tmp.c\_str()));

tmp = "";

}

if (IsOperations(postfix[i]) == true)

{

tmp1 = value.Pop();

tmp2 = value.Pop();

switch (postfix[i])

{

case '+':

value.Push(tmp1 + tmp2);

break;

case '-':

value.Push(tmp2 - tmp1);

break;

case '\*':

value.Push(tmp1\*tmp2);

break;

case'/':

value.Push(tmp2 / tmp1);

break;

}

}

}

return value.Pop();

}

intTPostfix::Compare(char tmp1)

{

if (tmp1 == '\*' || tmp1 == '/')

return 2;

if (tmp1 == '-' || tmp1 == '+')

return 1;

if (tmp1 == '(')

return 0;

}

bool TPostfix::CorrectRecord()

{

intlb = 0, rb = 0;

int operations = 0, operands = 0;

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

{

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

{

lb++;

continue;

}

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

{

rb++;

continue;

}

if (rb>lb)

return false;

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

operations++;

else

{

while (isalpha(infix[i + 1]) || isdigit(infix[i + 1]))

{

i++;

}

operands++;

}

}

if ((IsOperations(infix[0]) == true || IsOperations(infix[infix.size() - 1]) == true))

return false;

if ((operands == operations + 1) && (lb == rb))

return true;

else return false;

}

bool TPostfix::IsOperations(char s)

{

if (s == '+' || s == '-' || s == '\*' || s == '/')

return true;

return false;

}

bool TPostfix::IsOperand(char s)

{

if ((int(s) >= 65 &&int(s) <= 90) || (int(s) >= 97 &&int(s) <= 122))

return true;

return false;

}

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

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

constintMaxStackSize = 100;

template <class Telem>

class TStack

{

Telem \*pMem;

int size;

int top;

public:

TStack(int \_size)

{

size = \_size;

top = -1;

if ((size < 1) || (size >MaxStackSize))

throw size;

pMem = new Telem[size];

}

~TStack()

{

delete[] pMem;

}

bool IsFull()

{

if (top == (size - 1))

return true;

return false;

}

bool IsEmpty()

{

if (top == -1)

return true;

return false;

}

void Push(constTelem&val)

{

if (IsFull() == false)

{

top++;

pMem[top] = val;

}

else throw "error";

}

TelemPop()

{

if (IsEmpty() == false)

return pMem[top--];

else

throw "error";

}

TelemBack()

{

returnpMem[top];

}

};

#endif