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

ВЫЧИСЛЕНИЕ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

Выполнил: студент группы 381808-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шульман Е. А.

Подпись

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панов А. А.

Подпись

Нижний Новгород

2019

Содержание

[**Введение** 2](#_Toc24405246)

[**Постановка задачи** 3](#_Toc24405247)

[**Руководство пользователя** 4](#_Toc24405248)

[**Руководство программиста** 6](#_Toc24405249)

[Описание структуры программы 7](#_Toc24405250)

[Описание структуры данных 7](#_Toc24405251)

[**Заключение** 11](#_Toc24405252)

[**Литература** 12](#_Toc24405253)

[**Приложения** 13](#_Toc24405254)

[Приложение 1 13](#_Toc24405255)

[Приложение 2 14](#_Toc24405256)

[Фрагменты исходного кода программы 14](#_Toc24405257)

[Класс TPostfix 14](#_Toc24405258)

**Введение**

Одной из главных причин, лежащих в основе появления языков программирования высокого уровня, явились вычислительные задачи, требующие больших объёмов рутинных вычислений. Поэтому к языкам программирования предъявлялись требования максимального приближения формы записи вычислений к естественному языку математики. В этой связи одной из первых областей системного программирования сформировалось исследование способов выражений. Здесь получены многочисленные результаты, однако наибольшее распространение получил метод трансляции с помощью обратной польской записи , которую предложил польский математик Я.Лукашевич.

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.

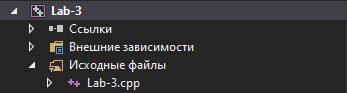
**Постановка задачи**

В рамках лабораторной работы ставится задача разработки программных средств, производящих обработку арифметических выражений, включая проверку правильности записи выражения, перевод в постфиксную форму и вычисление результата.

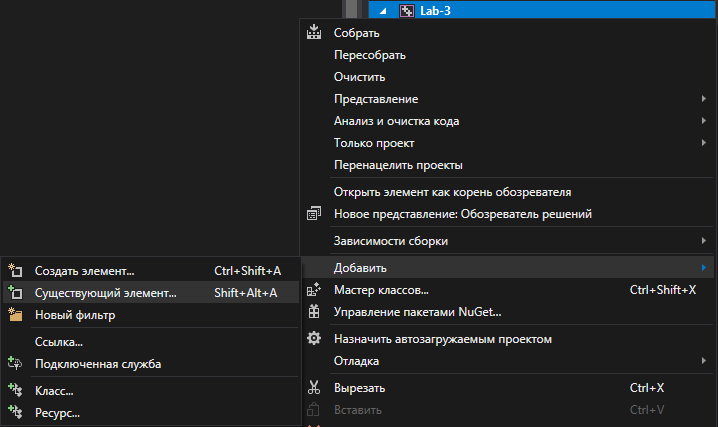
**Руководство пользователя**

По результатам работы был реализован class TPostfix для того, чтобы им пользоваться необходимо:

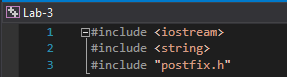
1. Создать новый проект.



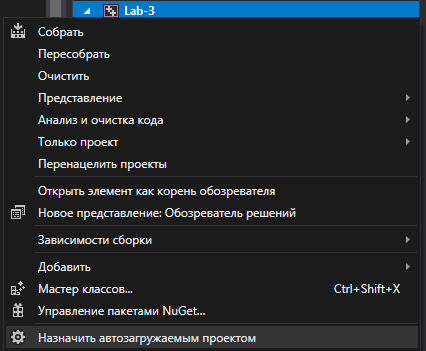
1. Добавляем существующие элементы и выбираем postfix.h и postfix.cpp.



1. Подключаем заголовочный файл.

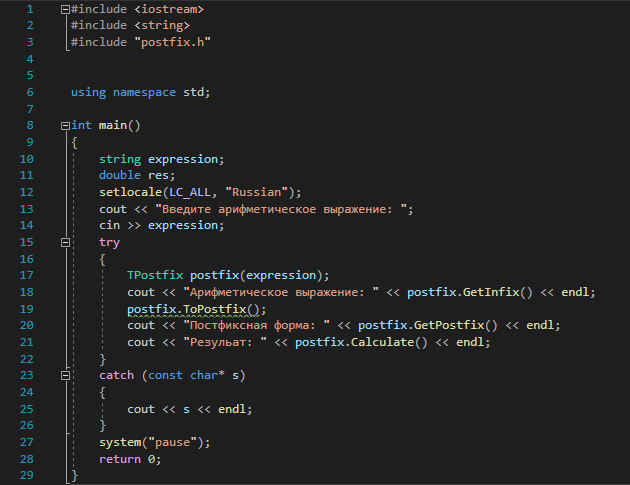
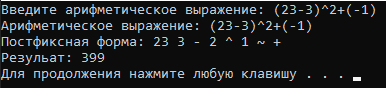


1. Назначаем наш проект автозагружаемым.



1. Теперь нашему проекту доступен класс TPostfix.

Проверим это:

**Руководство программиста**

Описание структуры программы

Составляющие проекта Lab-3:

* Lab-3 – приложение, вычисляющее арифметические выражения;
* Lab-3.cpp – точка старта программы;
* postfix.h – заголовочный файл класса TPostfix;
* postfix.cpp – исполняемый файл класса TPostfix;

Описание структуры данных

В данной лабораторной работе, мы получаем на вход арифметическое выражение в инфиксной форме.

Наша задача перевести арифметическое выражение из инфиксной формы записи в постфиксную и подсчитать результат.

Отличительной особенностью обратной польской нотации является то, что:

* Все аргументы (или операнды) расположены перед знаком операции.
* Запись набора операций состоит из последовательности операндов и знаков операций. Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.
* Результатом вычисления выражения становится результат последней вычисленной операции.

Алгоритм перевода из инфиксной формы записи в постфиксную:

1. Проверяем выражение на правильность скобок и операторов.
2. Проходимся по всему выражению и если находим унарный минус заменяем символ ‘-‘ на ‘~’.
3. В цикле начинаем посимвольно считывать выражение.

3.1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.  
3.2. Если символ - знак операции (+, -, \*, /, ^), то проверяем приоритет данной операции. Операция унарный минус имеет наивысший приоритет (допустим он равен 5). Операция возведение в степень имеет меньший приоритет (допустим он равен 4). Операции умножения и деления имеют меньший приоритет (равен 3). Операции сложения и вычитания имеют еще более меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.  
Получив один из этих символов, мы должны проверить стек:

а) Если стек все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.  
б) Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).

3.3. Если текущий символ - открывающая скобка, то помещаем ее в стек.  
3.4. Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

3.5.Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

Описание алгоритмов

Класс TPostfix

Наиболее сложные методы класса

* TPostfix (string tmp = "");

Конструктор создает объект класса с инфиксной формулой tmp и пустой постфиксной формулой.

* TPostfix (TPostfix& v);

Конструктор копирования создает новый объект класса, копируя данные из существующего v.

* bool IsOperator(char v);

Данный метод проверяет, является ли v символом оператора.

* bool IsNumeral(char v);

Данный метод проверяет, является ли v цифрой.

* bool CheckOperations();

Данный метод проверяет, правильность использования операторов.

* bool CheckBrackets();

Данный метод проверяет, правильность расставленных скобок.

* int Priority(char a, char b);

Возвращает приоритет между операторами a и b.

* string ToPostfix();

Основная функция класса! Переводит формулу из инфиксной формы записи в постфиксную.

* double Calculate();

Подсчитает значение формулы по постфиксной записи.

* string GetInfix();

Возвращает инфиксную форму записи.

* string GetInfix();

Возвращает постфиксную форму записи.

**Заключение**

Целью работы была реализация структуры данных, вычисляющая значение арифметических выражений.

В результате была написана структура данных, которая проверяет введённое арифметическое выражение, переводит арифметическое выражение из инфиксной формы записи в постфиксную и подсчитывает результат.

Программные средства созданы при помощи среды программирования С++ и содержат:

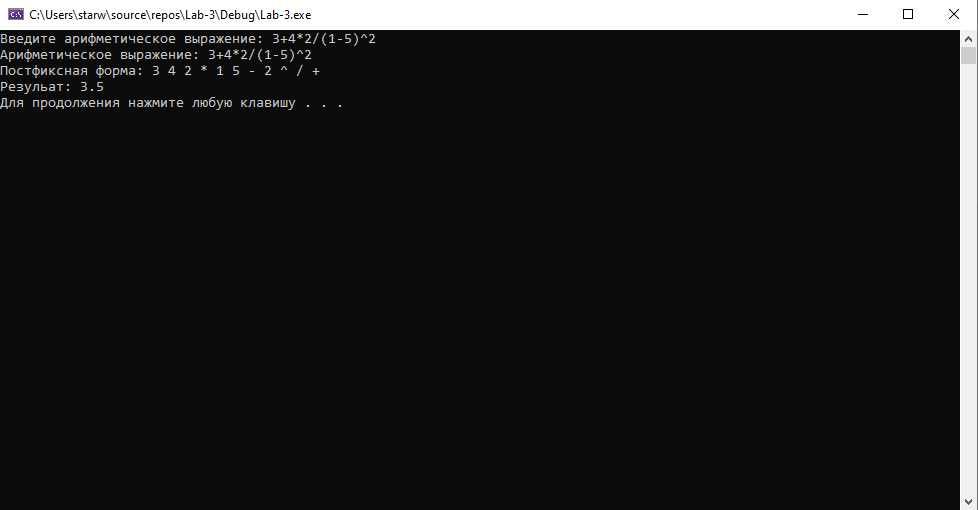
* класс TPostfix

В программе также реализована система обработки ошибок.

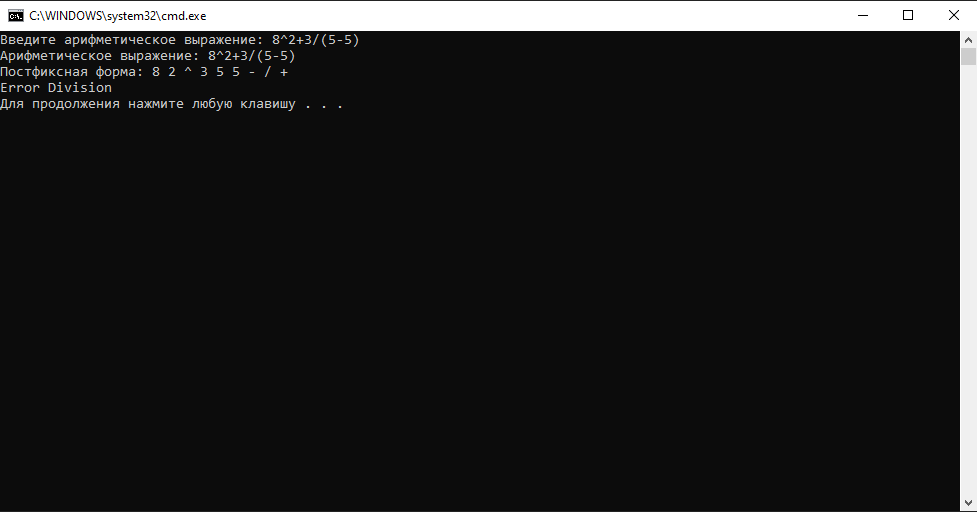
**Литература**

**Приложения**

Приложение 1



Приложение 2



## Фрагменты исходного кода программы

Класс TPostfix

bool TPostfix::IsOperator(char v)

{

if (v == '~' || v == '-' || v == '+' || v == '/' || v == '\*' || v == '^' || v == '(' || v == ')')

return true;

else

return false;

}

bool TPostfix::IsNumeral(char v)

{

if (int(v) >= 48 && int(v) <= 57)

return true;

return false;

}

bool TPostfix::CheckOperations()

{

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

if(!IsNumeral(infix[i]) && !IsOperator(infix[i]))

return false;

int Operands = 0, Operations = 0;

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

{

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

{

i++;

if (i == infix.size())

break;

}

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

Operations++;

if (!IsOperator(infix[i]))

{

while (!IsOperator(infix[i]))

{

i++;

if (i == infix.size())

break;

}

i--;

Operands++;

}

}

if (Operands == Operations + 1)

return true;

else return false;

}

bool TPostfix::CheckBrackets()

{

int LeftBrackets = 0, RightBrackets = 0;

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

{

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

LeftBrackets++;

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

RightBrackets++;

if (RightBrackets > LeftBrackets)

throw "Error Brackets";

}

if (LeftBrackets == RightBrackets)

return true;

else return false;

}

int TPostfix::Priority(char a, char b)

{

if (a == '~')

return 1;

if (b == '~')

return -1;

if (a == '^' && b != '~')

return 1;

if (b == '^' && a != '~')

return -1;

if ((a == '\*' || a == '/') && (b == '\*' || b == '/') || (a == '+' || a == '-') && (b == '+' || b == '-'))

return 0;

if ((a == '\*' || a == '/') && (b == '+' || b == '-'))

return 1;

if ((a == '+' || a == '-') && (b == '\*' || b == '/'))

return -1;

throw "not operation";

}

void TPostfix::SplitStr(string& str)

{

if (str.empty())

return;

if (str.back() != ' ')

str += ' ';

}

string TPostfix::ToPostfix()

{

if (!CheckBrackets())

throw "Error Brackets";

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

{

if (infix[0] == '-')

infix[0] = '~';

if (infix[j] == '-' && (infix[j-1] == '(' || infix[j - 1] == '~'))

infix[j] = '~';

}

int count = -1;

stack<char> Oper;

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

{

if (IsOperator(infix[i]))

{

if ((Oper.empty() || infix[i] == '(') && infix[i]!='~')

{

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

SplitStr(postfix);

Oper.push(infix[i]);

count++;

continue;

}

else if ((Oper.empty() || Oper.top() == '~') && infix[i] == '~')

{

Oper.push(infix[i]);

count++;

continue;

}

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

{

while (Oper.top() != '(')

{

SplitStr(postfix);

postfix += Oper.top();

Oper.pop();

count--;

}

Oper.pop();

count--;

}

else if ((Oper.top()) == '(' || Priority(infix[i], Oper.top()) == 1)

{

Oper.push(infix[i]);

SplitStr(postfix);

count++;

}

else if (Priority(infix[i], Oper.top()) == -1)

{

SplitStr(postfix);

while (!(Oper.empty()))

{

if (Oper.top() == '(')

break;

postfix += Oper.top();

Oper.pop();

SplitStr(postfix);

count--;

}

Oper.push(infix[i]);

count++;

}

else if (Priority(infix[i], Oper.top()) == 0)

{

SplitStr(postfix);

postfix += Oper.top();

Oper.pop();

Oper.push(infix[i]);

SplitStr(postfix);

}

}

else

postfix += infix[i];

if (i == infix.size() - 1)

{

for (count; count >= 0; count--)

{

SplitStr(postfix);

postfix += Oper.top();

Oper.pop();

}

}

}

return postfix;

}

double TPostfix::Calculate()

{

if (!CheckOperations() || postfix == "")

throw "Error Operand";

stack<double> Res;

string str;

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

{

if (postfix[i] == ' ')

continue;

if (!IsOperator(postfix[i]))

{

while (postfix[i] != ' ')

{

str += postfix[i];

i++;

}

Res.push(atof(str.c\_str()));

str = "";

}

else

{

if (postfix[i] == '~')

{

double a = Res.top();

Res.pop();

Res.push(-a);

continue;

}

double a = Res.top();

Res.pop();

double b = Res.top();

Res.pop();

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

Res.push(b + a);

else if (postfix[i] == '-')

Res.push(b - a);

else if (postfix[i] == '\*')

Res.push(b \* a);

else if (postfix[i] == '/')

{

if (a == 0.0)

throw "Error Division";

Res.push(b / a);

}

else if (postfix[i] == '^')

Res.push(pow(b,a));

}

}

return Res.top();

}