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 381808-1

Чесноков А.А.

**Проверил**:

доцент кафедры МОСТ, к.т.н.,

Сысоев А.В.

Нижний Новгород

2019

Содержание

[Введение 3](#_Toc529541653)

[1. Постановка задачи 4](#_Toc529541654)

[2. Руководство пользователя 5](#_Toc529541655)

[3.1. Описание структуры программы 6](#_Toc529541656)

[3.2. Описание алгоритмов 6](#_Toc529541657)

[4. Результаты экспериментов 7](#_Toc529541658)

[Заключение 8](#_Toc529541659)

[Литература 9](#_Toc529541660)

[Приложение 10](#_Toc529541661)

# Введение

В этой лабораторной работе рассматривается вычисление арифметических выражений. Одним из удобных способов их вычисления является перевод в постфиксную форму записи, так как она просто указывает порядок выполнения операций, кроме того не требует скобок. Знаки операций следуют сразу после операндов. Для вычисления выражений, записанных в постфиксной форме, удобно использовать такую структуру данных как стек, при этом он пригодится и для перевода в данную форму.

# Постановка задачи

Задача заключается в анализе введённой пользователем строки (предполагается арифметическое выражение в инфиксной форме): проверка на корректность ввода (отсутствие деления на ноль, верность скобок и пр.), замена введённых переменных на соответствующие числовые значения, перевод в польскую нотацию и непосредственно само вычисления и вывод значения выражения. Пользователю доступны основные арифметические операции (сложение, вычитание, умножение, деление, унарный минус), ввод переменных и констант. При этом пользователь может вводить выражение с любым количеством разделяющих пробелов, а имена переменных в любом регистре. В введённой строке убираются лишние пробелы, а все имена переменных переводятся в верхний регистр.

# Руководство пользователя

1. Запустить приложение.
2. В открывшейся консоли (Рис. 1) ввести арифметическое выражение, при некорректном вводе приложение уведомит вас об этом и предложит ввести его заново.
3. При успешном вводе выведется отформатированное выражение в инфиксной и постфиксной форме.
4. При наличии переменных ввести их числовые значения.
5. При успешном вычислении выведется результат и будет предложено выйти из приложения (введя q) или продолжить вычисление выражений, вернувшись к пункту 2.

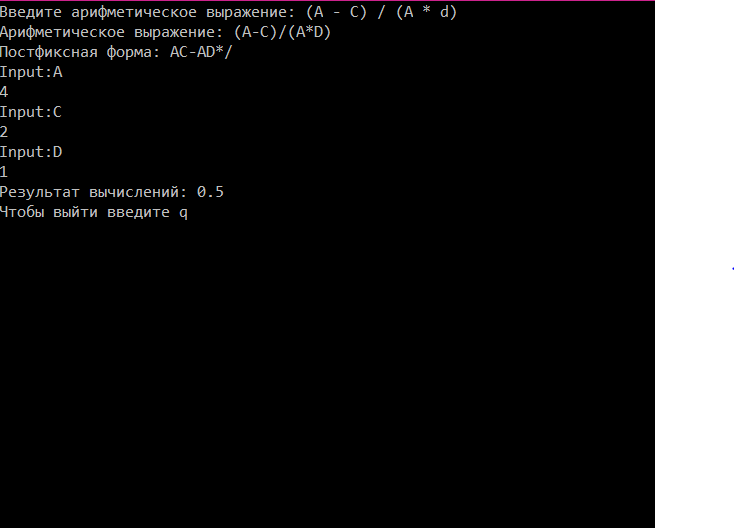


Рис. 1. Пример использования приложения

1. **Руководство программиста**

## Описание структуры программы

Программа состоит из двух заголовочных файлов (postfix.h, stack.h) и двух файлов с исходным кодом (postfix.cpp, sample\_postfix.cpp, методы класса стека реализованы в заголовочном файле, так как класс является шаблонным). Класс TStack реализует стек – его создание, добавление элемента в конец, выталкивание элемента с конца. Класс TPostfix реализует перевод из инфиксной формы записи в постфиксную, так же позволяет вычислять значения данного выражения.

## Описание алгоритмов

**Алгоритм перевода в Польскую нотацию:**

1. Введём приоритет для операций (умножение и деление – самый высокий, сложение вычитание – средний, скобки для компоновки подвыражений – низкий).
2. Для обработки создаем два стека.
3. Просматриваем посимвольно поступающую строку.
   1. Если увидели переменную перемещаем ее в первый стек, если ‘(‘ – во второй.
   2. Если встретили ‘)’ перемещаем все символы из второго стека в первый пока не встретим ‘(‘.
   3. Если встретили операцию и её приоритет не ниже, чем у операции на вершине второго стека, то кладём во второй стек.
   4. Иначе перекладываем из второго стека все операции с более высоким или равным приоритетом.
4. Когда символы в строке закончились переложить содержимое второго стека в первый.
5. Из первого стека переложить все символы в строку.

**Вычисление выражения:**

1. Создаем новый стек.
2. По порядку просматриваем строку.
   1. Если встретили переменную, то соответствующее ей значение кладём в стек.
   2. Если операцию – вынимаем из стека два верхних значения, выполняем данную операцию над ними и полученное значение кладем обратно в стек.
3. После просмотра всей строки в стеке будет лежать одно значение – результат вычислений.

# Результаты экспериментов

**Проведённые эксперименты:**

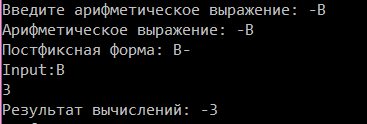
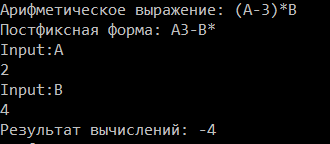
1. Переменные в разных регистрах и с произвольным числом пробелов корректно считываются (Рис. 2).
2. Применение унарного минуса (Рис. 3).
3. Некорректные скобки (Рис. 4).
4. Применение констант в выражении (Рис. 5).

Рис. 5. Результат эксперимента 4

Рис. 4. Результат эксперимента 3

Рис. 3. Результат эксперимента 2

Рис. 2. Результат эксперимента 1

# Заключение

В ходе выполнения лабораторной работы была реализована программа, вычисляющая значения арифметических выражений на основе стека. При этом для их ввода предусмотрена удобная для человека форма записи, а для вычислений – удобная для компьютера постфиксная форма записи. Дополнительной задачей была реализация собственного класса стека.

# Литература

1. Гергель, В.П. Методы программирования 1. Программа общего курса и описания лабораторных работ. Часть 1— М.: Н.Новгород: ННГУ, 1999, — 151с.

2. Холмс С. Стек MEAN. Mongo, Express, Angular, Node—Питер, 2017, 496 с.

3. Кормен, Т., Лейзерсон, Ч., Ривест, Р. Алгоритмы: построение и анализ - М.: МЦНМО, 2000.

# Приложение

1. Исходный код основной программы:

#include "postfix.h"

#include <iostream>

#include <string>

using namespace std;

int main()

{

string expression;

double res;

char c;

setlocale(LC\_ALL, "Russian");

start:

try

{

do

{

cout << "Введите арифметическое выражение: ";

getline(cin, expression);

TPostfix postfix(expression);

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

postfix.ToPostfix();

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

res = postfix.Calculate();

cout << "Результат вычислений: " << res << endl;

cout << "Чтобы выйти введите q" << endl;

cin >> c;

std::cin.ignore(32767, '\n');

} while (c != 'q');

}

catch (const Postfix\_Exception e)

{

switch (e)

{

case Postfix\_Exception::Invalid\_Infix:

cout << "Некорректный ввод\n";

goto start;

case Postfix\_Exception::Division\_By\_Zero:

cout << "Деление на ноль\n";

std::cin.ignore(32767, '\n');

goto start;

default:

break;

}

}

return 0;

}

1. Исходный код TStack:

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

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

enum class Stack\_Exception

{

Invalid\_Size,

Stack\_OverFlow,

Stack\_Empty

};

const int MaxStackSize = 100;

template <class T>

class TStack

{

T \*pMem;

int size;

int top;

public:

TStack(int \_size);

TStack(const TStack<T>& s);

void Push(T a);

T Pop();

bool IsFull() const;

bool IsEmpty() const;

~TStack()

{

delete [] pMem;

}

};

template<class T>

TStack<T>::TStack(int \_size)

{

size = \_size;

top = -1;

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

throw Stack\_Exception::Invalid\_Size;

pMem = new T[size];

}

template<class T>

TStack<T>::TStack(const TStack<T>& s)

{

size = s.size;

top = s.top;

pMem = new T[size];

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

{

pMem[i] = s.pMem[i];

}

}

template<class T>

void TStack<T>::Push(T a)

{

if (IsFull()) throw Stack\_Exception::Stack\_OverFlow;

top++;

pMem[top] = a;

}

template<class T>

T TStack<T>::Pop()

{

if (IsEmpty()) throw Stack\_Exception::Stack\_Empty;

return pMem[top--];

}

template<class T>

bool TStack<T>::IsFull() const

{

return top >= size - 1;

}

template<class T>

bool TStack<T>::IsEmpty() const

{

return top < 0;

}

#endif

1. Исходный код TPostfix:

#include "postfix.h"

#include "stack.h"

#include <algorithm>

#include <iostream>

int charToInt(const char& c)

{

return c - '0';

}

bool charIsVar(const char& c)

{

return (c >= 'A' && c <= 'Z');

}

bool charIsDigit(const char& c)

{

return (c >= '0' && c <= '9');

}

bool charIsVarOrDigit(const char& c)

{

return charIsDigit(c) || charIsVar(c);

}

TPostfix::TPostfix()

{

}

TPostfix::TPostfix(string s)

{

// удаление пробелов

s.erase(remove\_if(s.begin(), s.end(), isspace), s.end());

// проверка на корректность и перевод всех символов в заглавные

int count1 = 0;

int count2 = 0;

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

{

if (islower(s[i]))

{

s[i] = toupper(s[i]);

}

}

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

{

// проверка на допустимые символы

if (!(charIsVarOrDigit(s[i]) || s[i] == '(' || s[i] == ')'

|| s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/'))

throw Postfix\_Exception::Invalid\_Infix;

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

{

count1++;

// после открывающейся скобки не следует переменная или еще одна скобка

if (!(charIsVarOrDigit(s[i+1])) && s[i + 1] != '(') throw Postfix\_Exception::Invalid\_Infix;

}

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

{

//перед закрывающейся скобки не стоит переменная или еще одна скобка

if (!(charIsVarOrDigit(s[i - 1])) && s[i - 1] != ')') throw Postfix\_Exception::Invalid\_Infix;

count2++;

}

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

{

// после знака операции не следует переменная или открывающаяся скобка

if (!(charIsVarOrDigit(s[i + 1])) && s[i + 1] != '(') throw Postfix\_Exception::Invalid\_Infix;

if (s[i] != '-')

{

// перед знаком операции не следует переменная или закрывающаяся скобка

if (!(charIsVarOrDigit(s[i - 1])) && s[i - 1] != ')') throw Postfix\_Exception::Invalid\_Infix;

}

}

if (s[i] >= 'A' && s[i] <= 'Z')

{

// две переменные подряд

if (s[i + 1] >= 'A' && s[i + 1] <= 'Z')

{

throw Postfix\_Exception::Invalid\_Infix;

}

}

}

// простая проверка на валидность скобок

if (count1 != count2)

{

throw Postfix\_Exception::Invalid\_Infix;

}

infix = s;

}

string TPostfix::ToPostfix()

{

// максимальный размер стека это длина инфиксной строки

TStack<char> stack1(infix.length());

TStack<char> stack2(infix.length());

char top;

int m = infix.length();

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

{

if (charIsVarOrDigit(infix[i]))

{

stack1.Push(infix[i]);

continue;

}

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

{

stack2.Push(infix[i]);

continue;

}

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

{

while ((top = stack2.Pop()) != '(')

{

stack1.Push(top);

}

continue;

}

if (!stack2.IsEmpty())

{

top = stack2.Pop();

if (priority(infix[i]) > priority(top))

{

stack2.Push(top);

stack2.Push(infix[i]);

continue;

}

else

{

stack2.Push(top);

while (priority(top) >= priority(infix[i]))

{

stack2.Pop();

stack1.Push(top);

if (stack2.IsEmpty()) break;

top = stack2.Pop();

stack2.Push(top);

}

stack2.Push(infix[i]);

continue;

}

}

else

{

stack2.Push(infix[i]);

continue;

}

}

while (!stack2.IsEmpty())

{

char a = stack2.Pop();

stack1.Push(a);

}

postfix = "";

while (!stack1.IsEmpty())

{

char a = stack1.Pop();

postfix = a + postfix;

}

return postfix;

}

void TPostfix::SetNewInfix(string s)

{

TPostfix::TPostfix(s);

}

double TPostfix::Calculate()

{

map<char, double> values = read\_vars();

TStack<double> stack(postfix.length()); // максимальный размер стека - длина строки постфиксной формы

// должно хватить даже в самом худшем случае

int m = postfix.length();

double x, y, Result;

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

{

if (charIsVar(postfix[i]))

{

stack.Push(values[postfix[i]]);

}

if (charIsDigit(postfix[i]))

{

int tmp = charToInt(postfix[i]);

while (charIsDigit(postfix[i + 1]))

{

i++;

tmp \*= 10;

tmp += charToInt(postfix[i]);

}

stack.Push(tmp);

}

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

{

x = stack.Pop();

y = stack.Pop();

Result = y + x;

stack.Push(Result);

}

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

{

if (i == 1) stack.Push(-stack.Pop());

else

{

x = stack.Pop();

y = stack.Pop();

Result = y - x;

stack.Push(Result);

}

}

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

{

x = stack.Pop();

y = stack.Pop();

Result = y \* x;

stack.Push(Result);

}

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

{

x = stack.Pop();

y = stack.Pop();

if (x == 0)

{

throw Postfix\_Exception::Division\_By\_Zero;

}

Result = y / x;

stack.Push(Result);

}

}

Result = stack.Pop();

return Result;

}

int TPostfix::priority(char c)

{

switch (c)

{

case '(':

return 1;

break;

case '+':

return 2;

break;

case '-':

return 2;

break;

case '\*':

return 3;

break;

case '/':

return 3;

break;

}

return 0;

return 0;

}

map<char, double> TPostfix::read\_vars()

{

map<char, double> values;

double value;

for (const char c : postfix)

{

if ((c >= 'A') && (c <= 'Z'))

{

if (values.count(c) == 0)

{

cout << "Input:" << c << endl;

cin >> value;

values[c] = value;

}

}

}

return values;

}