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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Рябушева И.В.

Подпись

**Научный руководитель**:

Доцент каф.МОСТ

Кандидат технических наук

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

Подпись

Нижний Новгород

2018

**Оглавление**

[**Введение** 3](#_Toc530333855)

[Постановка задачи 4](#_Toc530333856)

[Руководство пользователя 5](#_Toc530333857)

[Руководство программиста 6](#_Toc530333858)

[Описание структуры программы 6](#_Toc530333859)

[Описание алгоритмов 6](#_Toc530333860)

[**Заключение** 7](#_Toc530333861)

[**Литература** 8](#_Toc530333862)

[**Приложение** 9](#_Toc530333863)

**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

В рамках лабораторной работы 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. В начальной – самой простой постановке – можно предполагать, что проверка записи выражения состоит в контроле правильности расстановки скобок, перевод в постфиксную форму производится только для корректных выражений, а вычисление – для корректных выражений, содержащих только числовые операнды и допустимые знаки операций.

# Руководство пользователя

Данная программа написана в среде разработки Microsoft Visual Studio 2017 на языке С++ и является консольным приложением. Пользователю предлагается создать переменную класса TPostfix передав параметром строку арифметического выражения. После ввода программа производит перевод выражения из инфиксной формы в постфиксную, а также производится вычисление выражения. В случае некорректного ввода программа выдаст ошибку.

Пример: TPostfix postfix ("(1+6/3\*5-4)\*(5-3)")

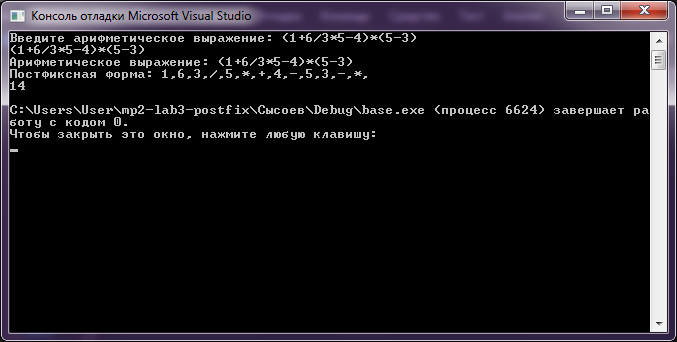


Рис. 1 Пример работы программы

# Руководство программиста

Программа написана при помощи двух классов TStack и TPostfix.

Класс TStack содержит три приватных поля:

1. T \*pMem – указатель шаблонного типа, отвечающий за хранения элементов стека;
2. int size – переменная отвечающая за размер стека;
3. int top - позиция последнего записанного элемента в стеке. По умолчанию top=-1.

Класс TStack содержит следующие методы:

1. TStack(int \_size = MaxStackSize) – конструктор класса;
2. ~TStack() – деструктор класса;
3. bool IsEmpty() – проверка на пустоту стека;
4. bool IsFull () - проверка на полноту стека;
5. T pop () – программа вытащит из стека верхний элемент, если стек пустой, то программа выдаст исключение;
6. void push (T v) – метод кладет в стек элемент, если он не полный, если стек заполнен, программа выдаст исключение;
7. T back() – возвращает значение последнего элемента стека, если стек пустой, то программа выдаст исключение.

Класс TPostfix содержит следующие приватные поля и методы:

1. string infix – переменная, содержащая в себе инфиксную форму выражения;
2. string postfix – переменная, содержащая в себе постфиксную форму выражения;
3. bool Operand(char op) – функция, проверяющая является ли символ, допустимым для вычисления, операндом;
4. bool IsOperation(char p) - функция, проверяющая является ли символ операцией;
5. int Priority(char a, char b) – функция для сравнения приоритетности операции.

Класс TPostfix содержит следующие методы в модификаторе доступа public:

1. TPostfix( string str = " ") – конструктор класса, на вход которого передается арифметическое выражение;
2. bool CheckInfix() – проверка корректности инфиксного выражения;
3. String GetInfix() – возвращает инфиксное выражение;
4. String GetPostfix() – возвращает постфиксное выражение;
5. string ToPostfix() – производит перевод инфиксного выражения в постфиксную форму;
6. double Calculate() – вычисляет арифметическое выражение.

# 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, производится перевод в постфиксную форму и вычисление выражения.

# Описание алгоритмов

**Алгоритмы класса TStack:**

TStack(int \_size = MaxStackSize) – конструктор принимающий значение длинны стека. В данном методе произойдет выделение динамической памяти и заполнение массива стека нулями.

bool IsEmpty() - метод, который возвращает тип bool. True если top равен -1.

bool IsFull () - метод, который возвращает тип bool. True если top равен (size-1).

T pop () – метод, который удаляет верхний элемент стека. Если стек не пуст, возвращает верхний элемент и уменьшает top на 1.

T back() – метод, который возвращает верхний элемент стека, если стек не пуст.

void push (T v) – метод, который добавляет значение в стек. Если стек не полон, увеличивает top и присваивает элементу массива pMem с индексом top переменную, которая пришла в метод в аргументе.

**Алгоритмы класса TPostfix:**

bool Operand(char op) – функция, проверяющая является ли символ, допустимым для вычисления, операндом, если op – операнд, метод вернет true, в противном случае - false.

bool IsOperation(char p) - функция, проверяющая является ли символ операцией, если р - операция, метод вернет true, в противном случае - false;

int Priority(char a, char b) - метод, который принимает два символа типа char, проверяет их отношение приоритетности, возвращает 1, если первый аргумент приоритетнее второго, 0, если приоритеты равны, -1, если второй приоритетнее первого.

String GetInfix() и String GetPostfix() - методы, которые возвращают инфиксную и постфиксную форму соответственно, типа string.

bool CheckInfix() – метод, проверяющий корректность строки infix. Возвращает тип bool. Происходит проверка на пустоту строки, является ли первый и последний элементы операторами и проверки на количество скобок и операторов.

string ToPostfix() – метод, преобразующий инфиксную форму в постфиксную и возвращающий строку, содержащую постфиксную форму. Алгоритм перевода в постфиксную запись обрабатывает исходный массив лексем и строит новый массив из тех же лексем, расположенных в другом порядке. Кроме того, необходим еще стек – аналогичный массив, используемый для временного хранения операций.

Алгоритм перевода выражения в постфиксную запись следующий.

1. Константы и переменные кладутся в формируемую запись в порядке их появления в исходном массиве.
2. При появлении операции в исходном массиве:
   1. если в стеке нет операций или верхним элементом стека является открывающая скобка, операции кладётся в стек;
   2. если новая операции имеет больший*[\*](http://natalia.appmat.ru/c&c++/postfisso.html" \l "nota)* приоритет, чем верхняя операции в стеке, то новая операции кладётся в стек;
   3. 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.
3. Открывающая скобка кладётся в стек.
4. Закрывающая скобка выталкивает из стека в формируемую запись все операции до ближайшей открывающей скобки, открывающая скобка удаляется из стека.
5. После того, как мы добрались до конца исходного выражения, операции, оставшиеся в стеке, перекладываются в формируемое выражение.

double Calculate() – метод, вычисляющий выражение используя постфиксную форму. Возвращает результат вычисления типа double.

Алгоритм следующий:

Просматриваем постфиксную запись. Значения констант кладутся в стек. Когда встречается операция, из стека берутся два верхних значения, вычисляется результат применения операции к этим значениям, и результат помещается в стек. Результатом вычисления является элемент, который остался в стеке в конце после просматривания постфиксной записи.

**Заключение**

В ходе данной лабораторной работы были реализована программа, позволяющая проверять корректность введенного пользователем инфиксного выражения, выполнять перевод введенного выражения в постфиксную форму и вычислять результат поученного выражения. Алгоритм перевода в постфиксную форму и последующего вычисления выражения основан на использовании стека. Реализация стека помогает наиболее рациональным и удобным способом реализовать алгоритмы.

**Литература**

1. Design Patterns: Elements of Reusable Object-Oriented Software («Приёмы объектно-ориентированного проектирования. Паттерны проектирования») Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес, ISBN 0-201-63361-2, 5-469-01136-4
2. Кнут Д. Э. Искусство программирования, том 4, A. Комбинаторные алгоритмы, часть 1 = The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1 / под ред. Ю. В. Козаченко. — 1. — Москва: Вильямс, 2013. — Т. 4. — 960 с. — ISBN 978-5-8459-1744-7
3. Бьерн Страуструп Язык программирования С++

**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)

{

size = \_size;

top = -1;

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

throw size;

pMem = new T[size];

}

~TStack()

{

delete [] pMem;

}

bool IsEmpty()

{

return top == -1;

}

bool IsFull()

{

return top == size - 1;

}

T pop()

{

if (IsEmpty())

throw "Error";

return pMem[top--];

}

void push(T v)

{

if (IsFull())

throw "Error";

pMem[++top] = v;

}

T back()

{

if (IsEmpty())

throw "Error";

return pMem[top];

}

};

#endif

**Модуль postfix.cpp**

#include "postfix.h"

#include "stack.h"

bool TPostfix::IsOperation(char p)

{

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

return true;

return false;

}

bool TPostfix::CheckInfix()

{

int a = 0, ops = 0;

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

{

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

ops++;

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

{

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

{

i++;

if (i == infix.size())

break;

}

i--;

a++;

}

}

if (a != ops + 1)

return false;

int left = 0, right = 0;

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

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

left++;

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

right++;

if (left != right)

return false;

if (infix[0] == ')' || infix[0] == '\*' || infix[0] == '/' || infix[0] == '-' || infix[0] == '+')

return false;

if (infix[infix.size() - 1] == '(' || infix[infix.size() - 1] == '\*' || infix[infix.size() - 1] == '/' || infix[infix.size() - 1] == '-' || infix[infix.size() - 1] == '+')

return false;

return true;

}

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

{

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

return 0;

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

return 1;

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

return -1;

if ((a == '+' || a == '-') && (b == '+' || b == '-'))

return 0;

throw "not operation";

}

bool TPostfix::Operand(char op)

{

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

if (op == i)

return true;

return false;

}

string TPostfix::ToPostfix()

{

if (!CheckInfix())

throw "Error";

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

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

{

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

{

int j = i;

while (!IsOperation(infix[j]) && j != infix.size() && infix[j] != ')')

{

postfix += infix[j];

j++;

}

i = j - 1;

postfix += ',';

}

else if (ops.IsEmpty())

ops.push(infix[i]);

else if (ops.back() == '(')

ops.push(infix[i]);

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

ops.push(infix[i]);

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

{

while (ops.back() != '(')

{

postfix += ops.pop();

postfix += ',';

}

ops.pop();

}

else if (Priority(infix[i], ops.back()) == 1)

ops.push(infix[i]);

else

{

while ((ops.back() != '('))

{

if (Priority(infix[i], ops.back()) == 1)

break;

postfix += ops.pop();

postfix += ',';

if (ops.IsEmpty())

break;

}

ops.push(infix[i]);

}

}

while (!ops.IsEmpty())

{

postfix += ops.pop();

postfix += ',';

}

return postfix;

}

double TPostfix::Calculate()

{

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

string TMP = "";

double tmp1, tmp2;

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

{

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

continue;

if (Operand(postfix[i]))

{

TMP = "";

TMP += postfix[i];

while (Operand(postfix[i + 1]) || (postfix[i + 1] == '.'))

{

TMP += postfix[i + 1];

i++;

}

stack.push(atof(TMP.c\_str()));

}

else

{

tmp1 = stack.pop();

tmp2 = stack.pop();

switch (postfix[i]) {

case'+':stack.push(tmp1 + tmp2); break;

case'-':stack.push(tmp2 - tmp1); break;

case'\*':stack.push(tmp1 \* tmp2); break;

case'/':stack.push(tmp2 / tmp1); break;

default: "Error";

}

}

}

return stack.pop();

}