МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**Вычисление арифметических выражений**

**Выполнила:** студентка группы

381703-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись

Ченкова А.А.

**Проверил:** ассистент кафедры

МОСТ ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Подпись

Волокитин В. Д.

Нижний Новгород  
2018

**Содержание**

[Введение 3](#_Toc532776775)

[1 .Постановка учебно-практической задачи 4](#_Toc532776776)

[2.Руководство пользователя 5](#_Toc532776777)

[3.Руководство программиста 6](#_Toc532776778)

[*Структура программы* 6](#_Toc532776779)

[*Структура данных* 6](#_Toc532776780)

[*Алгоритмы вычисления арифметических выражений* 8](#_Toc532776781)

[Заключение 11](#_Toc532776782)

[Литература 12](#_Toc532776783)

[Приложение 13](#_Toc532776785)

# 

# Введение

В жизни зачастую возникает необходимость вычислить то или иное арифметическое выражение. Некоторые, весьма простые, мы можем посчитать в уме, другие - на листке бумаги. Однако есть и такие выражения, которые можно посчитать, только потратив на это огромное количество времени. Для этого на помощь нам приходит компьютер. Персональные компьютеры уже давно не являются чем-то необычным, и мы активно используем их в повседневной жизни. А какой алгоритм даст нам наиболее быстрый результат и позволит считать большие выражения? Данный вопрос достаточно сложен, ведь алгоритмов можно придумать огромное количество. Один из них – вычисление арифметических выражений с использованием стека.

# 

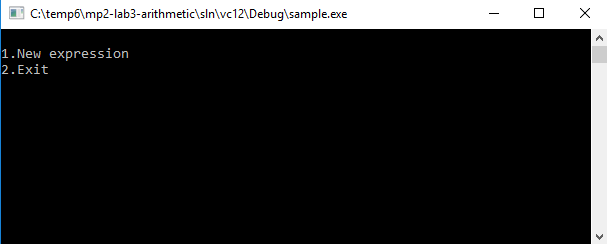
# 1 .Постановка учебно-практической задачи

*Формулировка задачи:*

Требуется разработать программу, выполняющую вычисление арифметического выражения с вещественными числами. Выражение в качестве операндов может содержать вещественные числа. Допустимые операции известны: +, -, /, \*. Допускается наличие знака "-" в начале выражения или после открывающей скобки. Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю вид ошибки.

# 2.Руководство пользователя

Необходимо запустить на выполнение файл “Sample.exe”, находящийся в папке “Debug”. Перед пользователем открывается прикладная программа.

Рис. 1. Главное меню программы

Для того чтобы ввести выражение, необходимо выбрать пункт 1. Для выхода из программы необходимо выбрать пункт 2.

После выбора пункта 1 пользователю необходимо ввести арифметическое выражение, используя цифры (целые и дробные) и операции: «+», «-», «\*», «/», «(» , «)». После выполнения расчетов программа выведет полученный результат и предложит повторный ввод или завершение работы программы. Если в выражении будет допущена синтаксическая ошибка, программа выведет вид ошибки пользователю и также предложит повторный ввод или выход.

# 

# 3.Руководство программиста

## *Структура программы*

Программа включает в себя следующие файлы:

* **Arithmetic.h** - содержит объявление класса «Lexem» и прототипы функций;
* **Arithmetic.cpp** – содержит реализацию методов класса «Lexem», а также функции перевода в обратную польскую запись, вычисления результата и поиска конкретных ошибок;
* **Stack.h** - содержит объявление класса «Stack» и реализацию его методов;
* **main\_arithmetic.cpp** - включает в себя прикладное приложение, демонстрирующее работу вышеназванных классов;
* **test\_stack.cpp** - содержит тесты для класса «Stack»;
* **test\_arithmetic.cpp** - содержит тесты для класса «Lexem», а также тесты для функций вычисления результата и поиска конкретных ошибок;
* **test\_main.cpp** - необходим для запуска Google tests.

## *Структура данных*

**Шаблонный класс Stack**

Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

const int MaxStackSize = 10; // Максимальный размер памяти для стека

Поля класса:

* + ValType \* stack; //указатель на массив
  + int end; //вершина стека
  + int n; //размер стека

Методы класса:

* Stack(int n1 = MaxStackSize); //конструктор (создает стек, указанного размера)
* ~Stack(); //деструктор
* ValType pop(); //извлечение элемента (из ячейки, содержащей последний добавленный элемент)
* void push(ValType val); //вставка элемента (в то место, куда указывает вершина стека, т.е. указатель на первый свободный элемент стека)
* bool IsEmpty(); //проверка на пустоту
* void Clean(); //очистка стека
* ValType Front(); //просмотр верхнего элемента (без удаления)
* int Size(); //получение количества элементов в стеке

**Класс Lexem**

Поля класса:

* double IsNumber; //переменная для хранения числа
* char IsOperation; //переменная для хранения операции
* int k; //переменная для определения типа лексемы (1-IsOperation, 2- IsNumber)

Методы класса:

* Lexem() {}; //конструктор без параметров
* ~Lexem() {}; //деструктор
* Lexem(char, int); // конструктор для операции
* Lexem(double, int); // конструктор для числа
* void set(char, int); //записывает значение лексемы для операции
* void set(double, int); //записывает значение лексемы для числа
* int sign(); //возвращает значение k
* double number(); //возвращает числовое значение лексемы
* char operation(); //возвращает тип операции лексемы

## *Алгоритмы вычисления арифметических выражений*

В программе были реализованы следующие алгоритмы:

1) Lexem\* PolishRecord(string stroka, int & k) - функция необходима для перевода выражения в обратную польскую запись.

1. Создаем массив лексем
2. С помощью цикла последовательно проверяем каждый элемент строки

* Если встречается цифра, то все элементы записываем в массив, до тех пор, пока не встретится арифметический знак
* Этот массив переводится в число (double), которое записывается в массив лексем (IsNumber=значение числа; k=2)
* Если встречается арифметический знак, то
* он записывается в стек (если до него не было других арифметических знаков)
* его приоритет сравнивается с приоритетом знака, расположенного на вершине стека (если до него были другие арифметические знаки)
* если его приоритет выше, то знак кладется в стек
* если его приоритет ниже или равен, то из стека извлекаются все знаки, чей приоритет выше или равен приоритета текущего. Эти знаки записываются в массив лексем, а текущий знак кладется в стек
* Если встречается скобка, то
* она кладется в стек (если эта скобка открывающая)
* из стека извлекаются все операции до открывающей скобки и в том числе она сама (если эта скобка закрывающая)

1. Осуществляем подсчет количества подряд идущих знаков (k). Последующее число умножаем на -1 k-ое число раз.

2)double result(Lexem \*x, int k)- функция необходима для вычисления выражения.

1. Идем в цикле по массиву лексем

* если лексема - число, то она кладется в стек
* если лексема – знак арифметической операции, то два последних числа из стека извлекаются, над ними выполняется текущая операция, а полученный результат опять складывается в стек

b. После разбора всей строки в стеке остается одно число, которое и будет результатом данного выражения. Это число функция возвращает в качестве результата.

3)bool errors(string stroka)- функция необходима для проверки корректности выражения.

1. В цикле все элементы строки последовательно сравниваем с допустимыми символами, если хотя бы один не совпадает с допустимым списком, то программа выводит сообщение об ошибке (The symbol is wrong)
2. С помощью цикла последовательно проверяем каждый элемент, складывая все открывающие скобки в специально созданный стек. После того, как встречается закрывающая скобка, открывающую извлекаем из стека. В случае если стек окажется пустым, программа выводит сообщение об ошибке (Brackets are wrong)
3. С помощью еще одного цикла осуществляем проверку арифметических знаков, которые идут друг за другом, а также проверку знака перед закрывающей скобкой. Если такие нашлись, то программа выдает сообщение об ошибке (Sequence of symbols is wrong). Исключение знак «-».
4. Функция также проверяет корректность начала (конца) выражения, которое по правилам может начинаться только с открывающей скобки, минуса или числа, а заканчиваться может только на закрывающую скобку или число. В противном случае программа выводит сообщение об ошибке (The expression starts with a wrong symbol (The expression ends with a wrong symbol))
5. Осуществляем проверку условия: перед открывающей скобкой может стоять открывающая скобка или арифметический знак. В противном случае программа выводит сообщение об ошибке (Sequence of symbols is wrong)
6. Осуществляем проверку условия: после закрывающей скобки может стоять еще одна закрывающая скобка или арифметический знак. В противном случае программа выводит сообщение об ошибке (Sequence of symbols is wrong)

# Заключение

В результате лабораторной работы была разработана программа, выполняющая вычисление арифметических выражений с вещественными числами. Данная работа позволила изучить структуру данных стек, а также алгоритм перевода заданной строки в вид обратной польской записи. Работа программы протестирована с помощью Google Tests.

# 

# Литература

Алгоритмы и методы: Обратная польская запись (исходники) [Электронный ресурс]. – Режим доступа: <http://www.interface.ru/home.asp?artId=1492>

# 

# Приложение

Lexem\* PolishRecord(string stroka, int & k) {

k = 0;

int a = 0;

int b = 0;

int с = 0;

char \*d;

double e;

Lexem \*x;

x = new Lexem[size(stroka)];

Stack<char> stc(size(stroka));

for (int i = 0; i < size(stroka); i++)

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

b = 0;

for (int j = i; (j < size(stroka)) && (stroka[j] != '+') && (stroka[j] != '-') && (stroka[j] != '\*') && (stroka[j] != '/') && (stroka[j] != '(') && (stroka[j] != ')'); j++)

b++;

d = new char[b + 1];

for (int j = i, n = 0; j < i + b; j++, n++)

d[n] = stroka[j];

d[b] = { '\0' };

e = atof(d);

for (int i = 1; i < с; i++)

e = -e;

с = 0;

x[k++].set(e, 2);

i += b - 1;

}

else {

if (a == 0) {

if ((stroka[i] == '-') && (i == 0)) {

с = 2;

}

else{

stc.push(stroka[i]);

a++;

с++;

}

}

else {

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

while (stc.Front() != '(') {

x[k++].set(stc.pop(), 1);

}

stc.pop();

}

else {

if ((stc.IsEmpty()==false) && ((priority(stc.Front()) < priority(stroka[i])) || (stroka[i] == '('))) {

if ((с == 0) || (stroka[i] == '('))

stc.push(stroka[i]);

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

с++;

}

else {

if (с == 0) {

while ((stc.IsEmpty()==false) && (stc.Front() != '(') ) {

x[k++].set(stc.pop(), 1);

}

stc.push(stroka[i]);

}

else

с++;

}

}

}

}

while (stc.IsEmpty()==false)

x[k++].set(stc.pop(), 1);

return x;

}

double result(Lexem \*x, int k) {

Stack <double> stc(k);

double res;

for (int i = 0; i < k; i++) {

if (x[i].sign() == 2) {

stc.push(x[i].number());

}

else {

double a = stc.pop();

double b = stc.pop();

switch (x[i].operation()){

case '+':

res = b + a;

break;

case '-':

res = b - a;

break;

case '\*':

res = b \* a;

break;

case '/':

if (a != 0)

res = b / a;

else throw 1;

break;

}

stc.push(res);

}

}

if (k == 1)

res = stc.pop();

return res;

}

bool errors(string stroka) {

bool a = true;

int b = 0;

Stack <char> stc(size(stroka));

int t = 0;

char br;

char c[17] = { '0','1','2','3','4','5','6','7','8','9','-','+','\*','/','.','(',')' };

for (int i = 0; i < size(stroka); i++) {

for (int j = 0; j < 17; j++)

if (stroka[i] == c[j]) {

b++;

}

if (b == 0) {

cout << endl << "The symbol is wrong";

a = false;

}

b = 0;

}

for (int i = 0; i < size(stroka); i++){

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

stc.push('(');

else

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

if (stc.IsEmpty()==false)

br = stc.pop();

else

t++;

}

}

if (t != 0) {

a = false;

cout << endl << "Brackets are wrong";

}

for (int j = 11; j < 15; j++)

if ((stroka[0] == c[j]) || (stroka[0] == ')')) {

a = false;

cout << endl << "The expression starts with a wrong symbol";

}

for (int j = 10; j < 16; j++)

if (stroka[size(stroka) - 1] == c[j]) {

a = false;

cout << endl << "The expression ends with a wrong symbol";

}

for (int i = 0; i < size(stroka); i++) {

for (int j = 10; j < 17; j++)

if (stroka[i] == c[j])

if ((i + 1) < size(stroka)) {

for (int l = 11; l < 15; l++)

if (((stroka[i + 1] == c[l]) || ((stroka[i + 1] == '-') && (stroka[i + 2]) == '(')) && (stroka[i] != ')')) {

a = false;

cout << endl << "Sequence of symbols is wrong";

}

}

}

for (int i = 1; i < size(stroka); i++) {

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

for (int j = 10; j < 14; j++)

if ((stroka[i - 1] == c[j]) || (stroka[i - 1] == '(')) {

b++;

}

if (b == 0) {

a = false;

cout << endl << "Sequence of symbols is wrong";

}

}b = 0;

}

for (int i = 0; i < size(stroka) - 1; i++) {

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

for (int j = 10; j < 14; j++)

if ((stroka[i + 1] ==c[j]) || (stroka[i + 1] == ')')) {

b++;

}

if (b == 0){

a = false;

cout << endl << "Sequence of symbols is wrong";

}

}

b = 0;

}

return a;

}