Министерство науки и высшего образования 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 382003-1

Шокуров Д. А.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Разработать программу, выполняющую вычисление арифметического выражения с вещественными числами. Выражение в качестве операндов может содержать переменные и вещественные числа. Допустимые операции известны: +, -, /, \*. Допускается наличие знака "-" в начале выражения или после открывающей скобки. Опционально - наличие математических функций (sin, соs, ln, exp, и т.д.). Программа должна выполнять предварительную проверку корректности выражения.

# Метод решения

Для решения данной задачи реализованы шаблонный класс my\_stack и класс arithmetic, в котором содержатся методы для перевода строки в обратную польскую форму записи (далее ОПЗ) и для подсчета значения выражения, записанного в ОПЗ.

ОПЗ - форма записи выражений, в которой операнды расположены перед знаками операций.

Класс arithmetic содержит основные методы:

1. prepare\_for\_parse() – сокращает последовательность подряд идущих минусов
2. parse() – перевод строки в ОПЗ
3. getAns() – подсчет значения выражения, записанного в ОПЗ

Для перевода выражения в ОПЗ воспользуемся следующим алгоритмом, реализованным в методе parse():

Вспомогательные переменные:

string temp – хранит необработанный фрагмент строки

my\_stack<pair<string, int>> stack – стек для хранения имени и приоритета операций

int extra\_pr – дополнительный приоритет к операциям

Считываем символ строки и добавляем его значение в переменную temp:

1. Если temp хранит число или точку, то добавляем temp к выходной строке
2. Если temp – открывающаяся скобка, то увеличиваем приоритет extra\_pr дальнейших операций
3. Если temp – закрывающаяся скобка, то уменьшаем приоритет extra\_pr дальнейших операций
4. Если temp – переменная, то добавляем ее имя к выходной строке
5. Если temp – операция, то перемещаем из стека все операции, приоритет которых больше или равен сумме собственного и дополнительного приоритета операции temp. Помещаем операцию temp в stack с приоритетом, равным сумме собственного и дополнительного приоритета extra\_pr.

В результате выполнения одного из пунктов будет выполнено соответствующее действие и строка temp будет очищена. Если строка temp не обработана, т.е. ее содержимое не подходит ни под одно из вышеперечисленных условий, тогда продолжаем выполнение, пока содержимое не будет распознано. В конце работы алгоритма содержимое стека операций будет перемещено в выходную строку.

Для того, чтобы подсчитать значение выражения, записанного в ОПЗ, воспользуемся следующим алгоритмом, реализованным в методе getAns():

Вспомогательные переменные:

my\_stack<double> stack

string temp

Считываем символ строки и добавляем его значение в переменную temp:

1. Если temp хранит операнд, то он помещается в стек
2. Если temp хранит операцию, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления. Результат выполненной операции кладётся на вершину стека.

Алгоритм продолжается до тех пор, пока полностью не обработан входной набор символов. После полной обработки набора символов результат лежит на вершине стека.

Программа выдает исключение и указывает на место встречи ошибки в случаях, когда:

1. не удалось распознать содержимое temp, т.е. введена некорректная команда
2. в числе присутствуют лишние точки
3. неверно расставлены скобки
4. пропущены операнды
5. пропущены операции

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

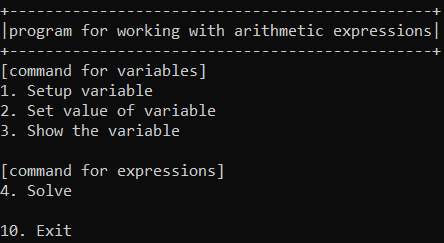
Возможны следующие операции со стеком:

1. вставка элемента
2. извлечение элемента
3. просмотр верхнего элемента
4. проверка на пустоту
5. получение количества элементов в стеке
6. очистка стека

# Руководство пользователя

1.1 Главное меню

Главное меню программы содержит 5 команд: 3 по работе с переменными, 1 для ввода и решения арифметического выражения и команду «Выход». Для выбора команды необходимо ввести в консоль её номер.

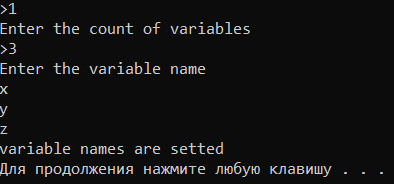


1.1 Главное меню

Далее приведено описание команд:

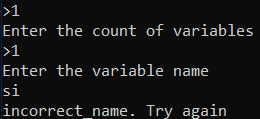
**Команды для переменных**

1) «Setup variable»: Для выбора введите 1. Позволяет настроить новый список переменных. Для этого необходимо указать кол-во переменных и их имена. В случае, если введенное имя является собственным префиксом какого-либо другого имени, зарезервированного программой, потребуется ввести новое имя.



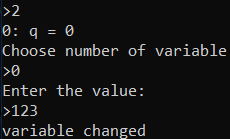
1.2 Окно настройки переменных

Пример некорректного ввода:



Так как программой предумотрено использование функции sin, то введенное название переменной некорректно.

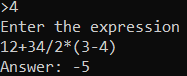
2) «Set value of variable»: Для выбора введите 2. Выводит список переменных и предлагает изменить значение одной из них. Для выбора переменной требуется ввести ее индекс.



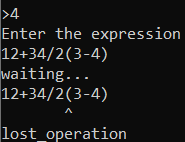
3) «Show the variable»: Для выбора введите 3. Выводит список доступных переменных и их значения. По умолчанию доступны переменные: x = 1, y = 2, z = 3

**Команды для выражений**

4) «Solve»: Для выбора введите 4. Ожидает ввода выражения. Если выражение корректно, то выводит ответ, иначе выводит сообщение о возникшей ошибке.



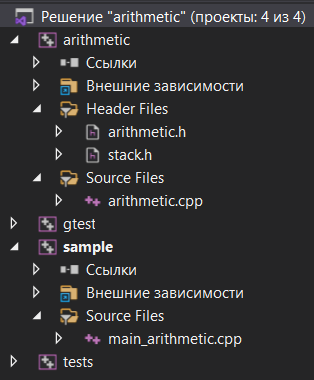
Пример сообщения об ошибке:



5) «Exit»: Для выбора введите 10. Завершает работу программы.

# Описание программной реализации

Программа разделена на файлы «arithmetic», «stack», «main\_arithmetic».



2.1 Решение

**main\_arithmetic.cpp**

В данном файле располагается реализация меню программы.

Глобальные переменные:

arithmetic ar – экземпляр класса для работы с арифметическими выражениями

const int Exit - код выхода из программы

Обработки ввода в консоль осуществляется в методе main(), который отображает главное меню программы и обрабатывает ввод консоли. При наличии команды, соответствующей введенному числу, данный метод вызывает ее, в противном случае выводит в консоль «command not found».

**void Initialize();**

Создание переменных x,y,z и инициализация их значениями.

Вызов дальнейших методов зависит от введенного в консоль числа.

1. **void change\_name\_of\_variable();**

Данная функция вызывается из main() в случае ввода пользователем 1. В данном методе осуществляется удаление старых и создание новых переменных с присваиванием им имен. В случае некорректного имени просит ввести новое.

1. **void show\_variable();**

Данная функция вызывается из main() в случае ввода пользователем 3. Выводит в консоль имена доступных переменных

1. **void change\_value();**

Данная функция вызывается из main() в случае ввода пользователем 2. Выводит в консоль имена доступных переменных. Просит ввести индекс переменной, значение которой будет изменено, и новое значение.

**arithmetic.cpp**

В данном файле хранится реализация класса arithmetic.

**Переменные:**

my\_vector<operation> op;

my\_vector<variable> val;

Внутри класса arithmetic содержатся структуры:

**operation**

Переменные структуры:

string def - определение операции

int priority - приоритет операции

int count\_arg - арность операции

double(\*f)(double, double) - указатель на функцию, отвечающую за выполнение операции

**variable**

**Переменные структуры:**

string def – название переменной

double value – значение переменной

**Методы:**

**double solve(string input)**

Внутри данного метода последовательно вызываются методы check\_name(), prepare\_for\_parse(), parse() и getAns().

**Check\_name()**

В данном методе осуществляется проверка на то, являются ли имена переменных префиксами имен операций и друг друга.

**string prepare\_for\_parse(string input)**

В данном методе осуществляется удаление лишних минусов из входной строки.

**string parse(string& input)**

В данном методе осуществляется перевод выражения в ОПЗ

**double arithmetic::getAns(string& input)**

В данном методе осуществляется подсчет значения выражения, записанного в ОПЗ

**stack.h**

В данном файле хранится реализация шаблонного класса stack.

Внутри класса stack содержится структура Node

**Переменные Node:**

Node<T>\* next - указатель на следующий элемент стека

T value - значение элемента стека

**Переменные класса stack:**

Node<T>\* top – указатель на вершину стека

size\_t count – кол-во элементов в стеке

**Методы класса stack:**

**bool empty()**

Данный метод возвращает true, если стек пуст, иначе false

**size\_t size()**

Данный метод возвращает размер стека.

**void push(const T& value)**

Данный метод принимает ссылку на экземпляр класса T. Копия этого объекта будет помещена на вершину стека.

**T pop()**

Данный метод возвращает верхний элемент и удаляет его из стека.

**T& back()**

Данный метод возвращает верхний элемент стека.

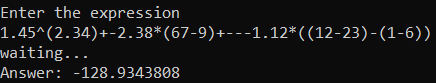
**void clear()**

Данный метод очищает стек.

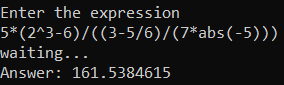
# Подтверждение корректности

Для подтверждения корректности к программе прилагается набор google testов, который проверяет корректность подсчета выражения на случайных примерах, включающих большинство арифметических операций, и на специальных примерах, например, верность подсчета выражения, включающего неассоциативные операции, такие как – и /. Корректные значения для выражений были посчитаны с помощью онлайн калькулятора. Выражение считается точно посчитанным, если погрешность составляет меньше . Также тесты рассматривают исключения, которые должны возникнуть при неверном наборе выражения.

Примеры:

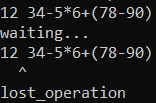
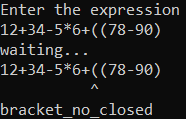
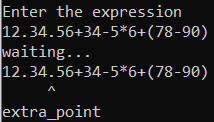


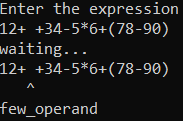
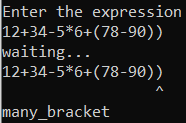
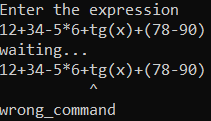
Верный ответ: -128.9343807790868932.



Верный ответ: 161.5384615384615385

Примеры ошибок:

# Заключение

Выполняя лабораторную работу я разработал программу, выполняющую вычисление арифметического выражения с вещественными числами.

# Приложение

//arithmetic.cpp

// реализация функций и классов для вычисления арифметических выражений

#include <string>

#include "..\include\arithmetic.h"

using namespace std;

typedef unsigned int uint;

void arithmetic::operation::set(string d, int p, int c, double(\*g)(double, double)) {

def = d;

priority = p;

count\_arg = c;

f = g;

}

arithmetic::arithmetic() {

op.push\_back(operation("~", -1, 0, [](double x, double y) {return 0.0; })); //конец строки

op.push\_back(operation("+", 1, 2, [](double x, double y) {return x + y; }));

op.push\_back(operation("-", 1, 2, [](double x, double y) {return x - y; }));

op.push\_back(operation("\*", 3, 2, [](double x, double y) {return x \* y; }));

op.push\_back(operation("/", 3, 2, [](double x, double y) {return x / y; }));

op.push\_back(operation("^", 4, 2, [](double x, double y) {return pow(x, y); }));

op.push\_back(operation("sin", 5, 1, [](double x, double y) {return sin(x); }));

op.push\_back(operation("cos", 5, 1, [](double x, double y) {return cos(x); }));

op.push\_back(operation("abs", 5, 1, [](double x, double y) {if (x >= 0)return x; else return -x; }));

op.push\_back(operation("`", 6, 1, [](double x, double y) {return -x; })); //унарный минус

}

bool arithmetic::is\_prefix(string& part, string& whole) {

if (part.size() > whole.size())return false;

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

if (part[i] != whole[i])return false;

}

return true;

}

bool arithmetic::name\_allowed(string& str) {

for (auto& it : str) {

if (('0' <= it && it <= '9') || it == '.')return false;

}

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

if (is\_prefix(str, op[i].def)) {

return false;

}

}

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

if (is\_prefix(str, val[i].def)) {

return false;

}

}

return true;

}

double arithmetic::solve(string &input) {

if (!check\_name())throw string("wrong variables");

input = prepare\_for\_parse(input);

string in = input;

in = parse(in);

return getAns(in);

}

bool arithmetic::check\_name() {

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

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

if (i != j && is\_prefix(val[i].def, val[j].def)) {

return false;

}

}

for (int k = 0; k < op.size(); k++) {

if (is\_prefix(val[i].def, op[k].def)) {

return false;

}

}

}

return true;

}

//delete extra minus

string arithmetic::prepare\_for\_parse(string input) {

string output = "";

char last = 0;

string temp = "";

if (input.size() != 0) {

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

char cur = input[i];

if (cur == '-' || (last == '-' && cur == ' ')) {

if (cur == '-') {

temp += cur;

}

}

else {

if (temp.size() != 0) {

if (temp.size() % 2 == 0) {

temp = "+";

}

else {

temp = "-";

}

output += temp;

temp = "";

};

output += cur;

}

if (cur != ' ')last = cur;

}

if (temp.size() != 0) {

if (temp.size() % 2 == 0) {

temp = "+";

}

else {

temp = "-";

}

output += temp;

temp = "";

};

}

return output;

};

string parseToString(uint number) {

string res = "";

do {

res += ((number % 10) + '0');

number /= 10;

} while (number > 0);

return res;

}

struct info {

string def;

int pr;

int index;

info() = default;

info(string d, int p, int i) {

def = d;

pr = p;

index = i;

}

};

void push\_operation(string& output, my\_stack<info>& stack, int priority) {

while (!stack.empty() && stack.back().pr >= priority) {

output += " " + parseToString((uint)stack.back().index) + "#" + stack.back().def;

stack.pop();

}

}

unsigned int parseToUInt(const string& str) {

uint res = 0;

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

if (!('0' <= str[i] && str[i] <= '9'))throw string("error: ParseToUInt()");

res \*= 10;

res += (str[i] - '0');

}

return res;

}

void operand(bool& was\_operand, bool able, int index) {

if (was\_operand && able) throw error("lost\_operation", index - 1);

was\_operand = able;

}

string arithmetic::parse(string input) {

my\_stack<info> stack;

string output = "";

string temp = "";

input += "~"; //конец строки, обладает отрицательным приоритетом

bool proc = false; //нужно ли очищать temp?

int extra\_pr = 0; //дополнительный приоритет к операциям

char last = 0; //последний символ

bool was\_point = false; //была ли уже точка при обработке числа double

int count\_bracket = 0; //кол-во скобок

bool unary\_minus = true; //может ли текущий минус быть унарным?

int place\_bracket = -1;

bool was\_operand = false;

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

temp += input[i];

proc = false;

if (('0' <= temp[0] && temp[0] <= '9') || temp[0] == '.') {

if (!(('0' <= last && last <= '9') || last == '.')) {

output += " " + parseToString(i) + "#";

was\_point = false;

operand(was\_operand, true, i);

}

if (temp[0] == '.') {

if (was\_point) {

throw error("extra\_point", i);

}

was\_point = true;

}

output += temp[0];

unary\_minus = false;

proc = true;

}

else if (temp[0] == '(' || temp[0] == ')') {

if (temp[0] == '(') {

extra\_pr += 100;

if (count\_bracket == 0) {

place\_bracket = i;

}

count\_bracket++;

unary\_minus = true;

}

else {

extra\_pr -= 100;

count\_bracket--;

unary\_minus = false;

}

if (count\_bracket < 0) {

throw error("many\_bracket", i);

}

proc = true;

}

else if (temp[0] == '-') {

if (unary\_minus) {

push\_operation(output, stack, 6 + extra\_pr);

stack.push({ "`",6 + extra\_pr ,i });

}

else {

push\_operation(output, stack, 1 + extra\_pr);

stack.push({ "-",1 + extra\_pr,i });

}

unary\_minus = true;

operand(was\_operand, false, i);

proc = true;

}

else if (temp[0] == ' ') {

proc = true;

}

else {

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

if (val[j].def == temp) {

output += " " + parseToString(i) + "#" + temp;

proc = true;

unary\_minus = false;

operand(was\_operand, true, i);

break;

}

}

if (!proc) {

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

if (op[j].def == temp) {

int cur\_pr = op[j].priority + extra\_pr;

push\_operation(output, stack, cur\_pr);

stack.push({ op[j].def, cur\_pr,i });

proc = true;

unary\_minus = true;

operand(was\_operand, false, i);

break;

}

}

}

}

if (proc) {

temp = "";

}

last = input[i];

}

if (temp.size() != 0) {

throw error("wrong\_command", input.size() - temp.size());

}

if (count\_bracket > 0) {

throw error("bracket\_no\_closed", place\_bracket);

}

return output;

};

double arithmetic::parseToDouble(const string& str) {

double res = 0.0;

double pos = str.size();

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

if (str[i] == '.') {

pos = i + 1;

}

else {

res \*= 10.0;

res += (str[i] - '0');

}

}

return res / pow(10.0, str.size() - pos);

}

int get\_index(const string& str) {

string temp = "";

int i = 0;

while (str[i] != '#') {

temp += str[i];

i++;

}

return parseToUInt(temp);

}

string get\_command(const string& str) {

string temp = "";

int i = 0;

while (i < str.size() && str[i] != '#') i++;

i++;

while (i < str.size()) {

temp += str[i];

i++;

}

return temp;

}

double arithmetic::getAns(string input) {

my\_stack<double> stack;

string str = "";

input += " ";

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

char& ch = input[i];

if (ch == ' ') {

if (str.size() != 0) {

int index = get\_index(str);

string temp = get\_command(str);

if ((('0' <= temp[0]) && (temp[0] <= '9')) || temp[0] == '.') {

stack.push(parseToDouble(temp));

}

bool flag = false;

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

if (temp == val[j].def) {

stack.push(val[j].value);

flag = true;

break;

}

}

if (!flag) {

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

if (temp == op[j].def) {

if (stack.size() < op[j].count\_arg) {

throw error("few\_operand", index + 1);

}

if (op[j].count\_arg == 1) {

double x = stack.pop();

double result = op[j].f(x, 0);

stack.push(result);

}

else if (op[j].count\_arg == 2) {

double y = stack.pop();

double x = stack.pop();

double result = op[j].f(x, y);

stack.push(result);

}

break;

}

}

}

}

str = "";

}

else {

str += ch;

}

}

return stack.pop();

}