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 381903-2

Гусев Никита Станиславович

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Руководство программиста 4](#_Toc26962563)

Описание алгоритмов.............................................................................................. 4

Описание структур данных..................................................................................... 5

[Руководство пользователя 6](#_Toc26962564)

[Подтверждение корректности 7](#_Toc26962566)

[Заключение 8](#_Toc26962568)

[Приложение 9](#_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. Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю вид ошибки и номера символов строки, в которых были найдены ошибки.

# Руководство программиста

**Описание алгоритмов**

На ввод подается строка string s, которая представляет собой само арифметическое выражение.

В программе реализованы алгоритмы определения символа в строке как оператора или операнда, установления приоритета операции, проверка корректности выражения, перевод в постфиксную запись, Google тесты.

**bool Error()** (ф-я проверки корректности выражения) Содержит в себе несколько циклов и условий, при нарушении которых будет выведено:

“string is empty”(строка пуста)

”wrong begin of expression”/”wrong end of expression”(в начале или в конце стоят недопустимые символы)

“spelling error” (нарушение в самой записи)

“several operations in a row” (записано несколько операций подряд)

“invalid symbol on (i) position” (введен некорректный символ)

“error in brackets” (ошибка в записи скобок).

**Определение символа как операнда или оператора**

bool IsOperator(char s) - Ф-я принимает на вход символ, а возвращает true или false, в зависимости от того является ли символ одним из: (+,-,\*,/,\_).

**Определение приоритета операции**

Int Priority(char s) – Ф-я принимает на вход символ, возвращает значение 2(\_), 1(\*,/), 0(+,-).

**Char IsUnar(int i)** – Ф-я проверяет, является ли минус унарным либо же бинарным. Возвращает “\_” или “-“.

**Перевод выражения в обратную польскую запись**

string NewInfix(string s)(меняет унарные минусы на “\_”)

string ToPostfix() – из инфиксной строки создается постфиксная строка

**Вычисление арифметического выражения**

double Calculator() – ф-я получает на вход постфиксную строку, возвращает значение.

**Описание структур данных**

В ходе лабораторной работы был создан класс Stack.

class Stack

{

TType \*Mass;//указатель на массив

int Top;//вершина стека

int Size;//размер стека

public:

Stack();//конструктор по умолчанию

Stack(int Size);//конструктор с параметром

void Push(TType val);//помещение элемента на вершину

TType Pop();//извлечение элемента с вершины стека

TType CheckTop();

TType GetSize();

void ClearStack();

bool IsEmpty();

bool IsFull();

void NewLen();

TType CheckTopEl();

~Stack();

};

Также был создан класс Arithmetic

class Arithmetic

{

string infix;

string postfix;

public:

Arithmetic(string s);

bool IsOperator(char s);

int Priority(char s);

char IsUnar(int i);

string NewInfix(string s);

string ToPostfix();

double count(double a, double b, char c);//вычисление выражения

double Unar(double a);//возвращает -a

double Calculator();//вычисление арифметического выражения

bool Error();//проверка

# Руководство пользователя

Для того, чтобы вычислить значение выражения, пользователю после запуска программы необходимо ввести само выражение после того, как выведется строка “Enter expression:”

Программа выведет результат при отсутствии ошибок, иначе сообщение об ошибке.

**Подтверждение корректности**

В ходе выполнения лабораторной работы реализованы гугл-тесты, а также булевая функция Error, которая проверяет введенное арифметическое выражение на корректность.

# Заключение

В ходе данной лабораторной работы была написана программа, которая вычисляет значение математического выражения с вещественными числами. Реализован шаблонный класс стек, класс Arithmetic, метод проверки, алгоритм перевода выражения в ОПЗ, алгоритм вычисления математического выражения, тестирование через google tests.

Программа выдает корректное значение арифметического выражения, либо сообщает об ошибке в веденной строке.

**Приложение**

class Arithmetic

{

string infix;

string postfix;

public:

Arithmetic(string s)

{

infix = s;

}

bool IsOperator(char s)

{

if (s == '+' || s == '\*' || s == '-' || s == '/' || s == '\_')

return true;

else return false;

}

int Priority(char s)

{

if (s == '\_')

return 2;

if (s == '\*')

return 1;

if (s == '/')

return 1;

if (s == '+')

return 0;

if (s == '-')

return 0;

}

char IsUnar(int i)

{

int j = i;

if ((j == 0) || infix[j - 1] == '(' || IsOperator(infix[j - 1]))

return '\_';

else return '-';

}

string NewInfix(string s)

{

string t;

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

{

if (!IsOperator(s[i]))

t += s[i];

else

if (IsUnar(i) == '\_')

t += '\_';

else t += s[i];

}

return t;

}

string ToPostfix()

{

Stack<char> opers(infix.size());

string tmp = NewInfix(infix);

infix = tmp;

int j = 0;

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

{

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

{

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

{

if (infix[i] != ' ')

{

int j = i;

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

{

postfix += infix[j];

j++;

}

i = j - 1;

postfix += ',';

}

}

else if (opers.IsEmpty() || opers.CheckTopEl() == '(')

opers.Push(infix[i]);

else if (Priority(infix[i]) >= Priority(opers.CheckTopEl()))

{

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

goto metka1;

else

opers.Push(infix[i]);

}

else if (Priority(infix[i]) < Priority(opers.CheckTopEl()))

{

while ((!opers.IsEmpty()) && (opers.CheckTopEl() != '(' || (Priority(opers.CheckTopEl()) < Priority(infix[i]))))

{

postfix += opers.Pop();

postfix += ',';

}

opers.Push(infix[i]);

}

else metka1:if (infix[i] == '(')

opers.Push(infix[i]);

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

{

while (opers.CheckTopEl() != '(')

{

postfix += opers.Pop();

postfix += ',';

}

opers.Pop();

}

}

while (!opers.IsEmpty())

{

char t = opers.Pop();

if (t != '(' && t != ')')

{

postfix += t;

postfix += ',';

}

}

return postfix;

}

}

double count(double a, double b, char c)

{

double res;

if (c == '+')

res = a + b;

if (c == '-')

res = a - b;

if (c == '\*')

res = a \* b;

if (c == '/')

{

if (b == 0) throw "impossible to divide by zero";

res = a / b;

}

return res;

}

double Unar(double a)

{

return (-a);

}

double Calculator()

{

double res, k;

Stack<double> nums(postfix.size());

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

{

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

goto metka;

if (!IsOperator(postfix[i]))

{

string tmp;

tmp += postfix[i];

i++;

while (postfix[i] != ',')

{

tmp += postfix[i];

i++;

}

nums.Push(atof(tmp.c\_str()));

}

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

{

double d = Unar(nums.Pop());

nums.Push(d);

i++;

}

else

{

k = nums.Pop();

res = count(nums.Pop(), k, postfix[i]);

nums.Push(res);

i++;

}

}

metka: return (nums.Pop());

}

bool Error()

{

bool res = true;

int l = 0;

Stack<char> x(infix.size());

int t = 0;

char brasc;

if (infix.empty())

{

cout << " string is empty" << endl;

return false;

}

else

{

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

{

res = false;

cout << "wrong begin of expression" << endl;

}

if (IsOperator(infix[infix.size() - 1]) || (infix[infix.size() - 1] == '(') || (infix[infix.size() - 1] == '.'))

{

res = false;

cout << "wrong end of expression" << endl;

}

}

for (int i = 0; i < infix.size() - 2; i++)

{

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

{

if (isdigit(infix[i + 1]) == 0)

{

cout << "spelling error" << endl;

res = false;

}

}

}

for (int i = 0; i < infix.size() - 2; i++)

{

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

{

if ((isdigit(infix[i + 1]) != 0) || (int(infix[i + 1]) >= 65 && int(infix[i + 1]) <= 90) || ((int(infix[i + 1]) >= 97 && int(infix[i + 1]) <= 122)) || (infix[i + 1] == '.'))

{

cout << "spelling error" << endl;

res = false;

}

}

}

for (int i = 0; i < infix.size() - 2; i++)

{

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

{

if (infix[i + 1] == '+' || infix[i + 1] == '\*' || infix[i + 1] == '/')

{

cout << "several operations in a row" << endl;

res = false;

}

}

}

for (int i = 0; i < infix.size() - 2; i++)

{

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

{

if ((infix[i + 1] == ')') || (infix[i + 1] == '.'))

{

cout << "spelling error" << endl;

res = false;

}

}

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

{

if (!((isdigit(infix[i]) != 0) || (infix[i] == ')') || (infix[i] == '(') || IsOperator(infix[i]) || (infix[i] == '.') || (infix[i] == ' ')))

{

cout << "invalid symbol on " << i + 1 << " position" << endl;

res = false;

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

x.Push('(');

else

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

{

if (!(x.IsEmpty()))

{

brasc = x.Pop();

}

else

{

t++;

}

}

}

if (t != 0)

{

res = false;

cout << "error in brackets" << endl;

}

return res;

}