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/ка группы 38хххх-х

Фамилия И.О.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2019

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

1. Разработка шаблонного класса Stack
2. Разбиение исходного арифметического выражения на лексемы (т.е. выделить операнды, операции, переменные)
3. Проверка корректности выражения:
4. Перевод выражения в постфиксную (польскую) запись
5. Вычисление выражения по постфиксной записи
6. Создание консольного интерфейса пользователя

# Метод решения

Для решения задачи был разработан шаблонный класс Stack со всем необходимым функционалом. Он используется для перевода в выражения в постфиксную форму и его вычисления. Класс Stack имеет три Private поля: data -для хранения данных, head - соответствующий вершине стека и n - размер стека. Имеются такие методы как: Конструктор(и в то же время конструктор по умолчанию), деструктор, “положить элемент в стек”, “достать последний элемент”, “стек полон?”, “стек пуст? ”, “показать стек ”.

Реализован класс Token для хранения лексем и перевода выражения в ОПЗ. Он описывается тремя полями: n (число), переменная типа std::string op для хранения лексемы, которая не является числом и переменная num типа bool, которая отвечает на вопрос “В лексеме хранится число?”. Реализованы методы: конструктор, конструктор по умолчанию, деструктор, “взять число”, “взять лексему”, “это число?”, “это строка?”, “получить приоритет”(для не численных лексем), перегружен оператор “=”.

Функция TranslateToRPE разбивает введенную пользователем строку на лексемы и кладёт её в соответствующий массив, попутно проверяя на возможные ошибки.

Перевод в ОПЗ:

На вход подаётся массив лексем. Начинаем цикл по массиву и смотрим, если взятая лексема число, то кладем в новый массив лексем, если операция, то смотрим на стек. Если стек пустой, кладём операцию в стек. В стеке уже есть операция – сравниваем приоритеты операций. Если приоритет операции уже лежавшей в стеке больше приоритета поступающей, то достаем ее и кладем в новый массив. После выполнения цикла кладем все операции из стека в конец нового массива.

Функция solve получает массив лексем в ОПЗ. Начинаем цикл по лексемам. Если очередная лексема – операнд, то кладём её в стек. Если бинарная операция – достаём два числа из стека и производим операцию и кладём результат в стек.

# Руководство пользователя

Ввести математическое выражения.

Правила ввода:

* Выражение вводится без пробелов
* Доступны такие переменные как x, y, z, k, n, и t
* Доступны математические функции sin(), cos(), ln(), exp()

# Описание программной реализации

Программа использует библиотеки “string”, cmath, stdexept, и iostream

В файле stack.h находится реализация шаблонного класса Stack

В файле arithmetic.h находится объявление и реализация класса Token и функции solve и translationToRPE,

Тесты расположены в папке test: test\_arithmetic.cpp, test\_main.cpp, test\_stack.cpp

Для решения задачи был разработан шаблонный класс Stack со всем необходимым функционалом. Он используется для перевода в выражения в постфиксную форму и его вычисления. Класс Stack имеет три Private поля: data -для хранения данных, head - соответствующий вершине стека и n - размер стека. Имеются такие методы как: Конструктор(и в то же время конструктор по умолчанию), деструктор, “положить элемент в стек”, “достать последний элемент”, “стек полон?”, “стек пуст? ”, “показать стек ”.

Реализован класс Token для хранения лексем и перевода выражения в ОПЗ. Он описывается тремя полями: n (число), переменная типа std::string op для хранения лексемы, которая не является числом и переменная num типа bool, которая отвечает на вопрос “В лексеме хранится число?”. Реализованы методы: конструктор, конструктор по умолчанию, деструктор, “взять число”, “взять лексему”, “это число?”, “это строка?”, “получить приоритет”(для не численных лексем), перегружен оператор “=”.

Остальные функции принимают непосредственное участие в переводе выражения в ОПЗ.

translationToRPE - На вход приходит строка с выражением. Далее запускаем цикл до конца строки и начинаем выделять лексемы, попутно проверяя на корректность.

В цикле первым делом проверяем элемент массива на то, является ли он цифрой, если да, то проверяем символы за ними, являются ли они продолжением числа (число может быть как типа int, так и double) и записываем полученное число в массив лексем соответствующим элементом.

Далее проверяем, не является следующая запись в выражении математической функцией (sin, cos, ln или exp). Также проверяем наличие скобок сразу после записи имени функции.

После проверяем, не является ли очередной символ переменной (в данной программе могут быть использованы только переменные x, y, z, k, n, t). Если да, то просим пользователя ввести значение текущей переменной и записываем его.

Далее проверяем, не является ли символ знаком арифметической операции (“+”, “-”, “ \* ”, “/”)

В конце проверяем, является элемент скобкой (открывающей или закрывающей). И записываем количество левых и правых скобок.

Если найдется элемент, не подходящий под один из вышеописанных случаев, значит выражение написано некорректно.

Далее идёт алгоритм перевода выражения в ОПЗ

Рассматриваем поочередно каждый символ:

1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.
2. Если символ - знак операции (“+”, “-”, “ \* ”, “/”), то проверяем приоритет данной операции.

а) Если стек все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.  
б) Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).

1. Если текущий символ - открывающая скобка, то помещаем ее в стек.
2. Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

Метод solve (решение выражения в ОПЗ)

Получаем массив лексем в ОПЗ и его размер. Далее действуем по алгоритму:

1. Обработка входного символа
   1. Если на вход подан операнд, он помещается на вершину стека.
   2. Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления. Результат выполненной операции кладётся на вершину стека.
   3. Если унарная операция, то из стека достаем только одно число.
2. Если входной набор символов обработан не полностью, перейти к шагу 1.
3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека.

# Подтверждение корректности

Для подтверждения корректности в программе были реализованы гугл тесты. В файле test\_stack.cpp реализованы простые тесты для проверки стека. В файле test\_arithmetic.cpp – тесты для проверки корректности перевода в ОПЗ. Проверяются: корректность программы при вводе одного числа, проверка несколькими корректными выражениями, а также проверка на создание исключения при попытке ввода некорректного приложения.

# Заключение

В ходе лабораторной работы была написана программа, которая способна принимать математическое выражение(с некоторыми математическими функциями), переводить это выражение в постфиксную форму и решать его. Реализованы: классы stack и token, а также функция перевода в ОПЗ.

# Приложение

double translationToRPE(std::string s)

{

int i = 0, j = 0;

int lBrackets = 0;

int rBrackets = 0;

Token\* tokens = new Token[s.length()];

double tmp, x, d;

struct variable {

bool avail = false;

double val;

};

struct variable sX;

struct variable sY;

struct variable sZ;

struct variable sK;

struct variable sN;

struct variable sT;

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

{

tokens[j++] = -1;

tokens[j++] = "\*";

i++;

}

while (s[i] != '\0') //конец строки

{

if ((s[i] >= '0') && (s[i] <= '9')) {

x = 0;

d = 10;

while ((s[i] >= '0') && (s[i] <= '9'))

{

tmp = (double)(s[i] - '0');

x = x \* 10 + tmp;

i++;

}

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

i++;

while ((s[i] >= '0') && (s[i] <= '9'))

{

if ((s[i] >= '0') && (s[i] <= '9')) {

tmp = (double)(s[i] - '0');

x = x + tmp / d;

i++;

d = d \* 10;

}

}

}

tokens[j] = x;

j++;

}

else if ((s[i] == 's') && (s[i + 1] == 'i') && (s[i + 2] == 'n'))

{

if (s[i + 3] != '(')

throw("Arguments of mathematical functions must be written in parentheses");

tokens[j++] = "sin";

i = i + 3;

}

else if ((s[i] == 'c') && (s[i + 1] == 'o') && (s[i + 2] == 's'))

{

if (s[i + 3] != '(')

throw("Arguments of mathematical functions must be written in parentheses");

tokens[j++] = "cos";

i = i + 3;

}

else if ((s[i] == 'l') && (s[i + 1] == 'n'))

{

if (s[i + 2] != '(')

throw("Arguments of mathematical functions must be written in parentheses");

tokens[j++] = "ln";

i = i + 2;

}

else if ((s[i] == 'e') && (s[i + 1] == 'x') && (s[i + 2] == 'p'))

{

if (s[i + 3] != '(')

throw("Arguments of mathematical functions must be written in parentheses");

tokens[j++] = "exp";

i = i + 3;

}

else if (s[i] == 'x') {

if (sX.avail == true)

tokens[j++] = sX.val;

else {

std::cout << "input " << s[i] << std::endl;

double value;

std::cin >> value;

sX.val = value;

sX.avail = true;

tokens[j++] = value;

}

}

else if (s[i] == 'y') {

if (sY.avail == true)

tokens[j++] = sY.val;

else {

std::cout << "input " << s[i] << std::endl;

double value;

std::cin >> value;

sY.val = value;

sY.avail = true;

tokens[j++] = value;

}

}

else if (s[i] == 'z') {

if (sZ.avail == true)

tokens[j++] = sZ.val;

else {

std::cout << "input " << s[i] << std::endl;

double value;

std::cin >> value;

sZ.val = value;

sZ.avail = true;

tokens[j++] = value;

}

}

else if (s[i] == 'k') {

if (sK.avail == true)

tokens[j++] = sK.val;

else {

std::cout << "input " << s[i] << std::endl;

double value;

std::cin >> value;

sK.val = value;

sK.avail = true;

tokens[j++] = value;

}

}

else if (s[i] == 'n') {

if (sN.avail == true)

tokens[j++] = sN.val;

else {

std::cout << "input " << s[i] << std::endl;

double value;

std::cin >> value;

sN.val = value;

sN.avail = true;

tokens[j++] = value;

}

}

else if (s[i] == 't') {

if (sT.avail == true)

tokens[j++] = sT.val;

else {

std::cout << "input " << s[i] << std::endl;

double value;

std::cin >> value;

sT.val = value;

sT.avail = true;

tokens[j++] = value;

}

}

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

{

tokens[j] = "+";

j++;

i++;

}

else if (s[i] == '\*')

{

tokens[j] = "\*";

j++;

i++;

}

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

{

tokens[j] = "/";

j++;

i++;

}

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

{

if (tokens[j - 1].isOp() && tokens[j - 1].getop() != ")")

{

tokens[j++] = "un-";

i++;

}

else

{

tokens[j] = "-";

j++;

i++;

}

}

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

{

if(s[i+1] == ')')

throw std::logic\_error("Error with brackets");

tokens[j] = "(";

j++;

i++;

lBrackets++;

}

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

{

if(s[i + 1] != '\0')

if((s[i + 1] != '+') && (s[i + 1] != '-') && (s[i + 1] != '\*') && (s[i + 1] != '/') & (s[i + 1] != ')'))

throw std::logic\_error("Invalid token after the brackets");

tokens[j] = ")";

j++;

i++;

rBrackets++;

}

else

{

throw ("ERROR");

}

}

if (lBrackets != rBrackets) {

throw ("Error with brackets");

}

Token\* arr; // перевод в ОПЗ

arr = new Token[j];

Stack<Token> t;

int l = 0;

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

{

if (tokens[k].isNum())

{

arr[l] = tokens[k];

l++;

}

else

if (t.IsEmpty())

{

t.push(tokens[k]);

}

else

{

if (tokens[k].getPriority() == 1)

{

t.push(tokens[k]);

}

else

{

while (!(t.IsEmpty()) && (tokens[k].getPriority() <= t.value().getPriority()))

{

if (t.value().getPriority() == 1)

{

t.pop();

break;

}

else

{

arr[l] = t.pop();

l++;

}

}

if (tokens[k].getPriority() != 0)

{

t.push(tokens[k]);

}

}

}

}

while (!(t.IsEmpty()))

{

arr[l] = t.pop();

l++;

}

return solve(arr, l);

}