**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

ОТЧЁТ

**по курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант 1.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Баязитов О.О. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

## Постановка задачи и описание реализуемого класса и методов.

Необходимо реализовать простейшую версию калькулятора. Пользователю должен быть доступен ввод математического выражения, состоящего из чисел и арифметических знаков. После программа переводит математическое выражение в формате обратной польской нотации, а так же выводит ответ.

## Обоснование выбора используемых структур данных.

Для реализации стэка был реализован однонаправленный список.

Был использован string для хранения выражения и для хранения отдельных функций. Функции в стеке так же в формате string.

## Описание алгоритма программы

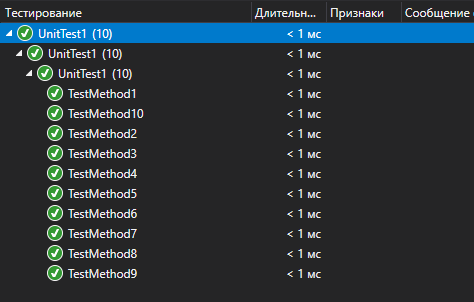
Выражение в инфиксной форме просматривается слева направо.

Если просматриваемый символ открывающая, то помещаем его с стек.

Если просматриваемый символ является оператором или функцией, то удаляем из стека все верхние операторы с приоритетом, большим или равным текущему и помещаем их в выходную строку. Затем помещаем его в стек. Если просматриваемый символ является закрывающей скобкой, то извлекаем из стека все операторы до открывающей скобки и добавляем их в выходную строку. При вычислении выражение просматривается в постфиксной форме слева направо. Если текущий элемент является операндом, то помещаем его в стек. Если текущий элемент является оператором, то достаем из стека одно или два числа в зависимости от операции.

## Реализация Unit-тестов

Все тесты пройдены успешно.



## Пример работы









## Листинг

Postfix.h

#pragma once

#include <string>

#include "Stack.h"

using namespace std;

class PostfixClass

{

public:

PostfixClass();

~PostfixClass();

void inputstring();

void ConvertToPostfix();

double Calculate();

int GetPriority(string s);

string outstring();

string infix;

string postfix;

private:

Stack stack;

};

Stack.h

#pragma once

#include <string>

using namespace std;

class Stack

{

private:

void reset\_list();

class StackNode

{

public:

StackNode(string data, StackNode\* next = nullptr, StackNode\* prev = nullptr) {

this->data = data;

this->next = next;

this->prev = prev;

}

~StackNode() {}

string data;

StackNode\* next;

StackNode\* prev;

};

StackNode\* head;

StackNode\* tail;

size\_t size;

public:

Stack();

~Stack();

void push\_back(string newElem);

string pop\_back();

string getTop();

size\_t get\_size();

void clear();

bool isEmpty();

};

Postfix.cpp

#include "Postfix.h"

#include <iostream>

#include <cmath>

const double PI = 3.14159;

const double E = 2.71828;

PostfixClass::PostfixClass()

{

}

PostfixClass::~PostfixClass()

{

}

void PostfixClass::inputstring()

{

std::getline(std::cin, infix);

}

int PostfixClass::GetPriority(string c)

{

if (c == "cos" || c == "sin" || c == "tg" || c == "ctg" || c == "log" || c == "ln" || c == "sqrt" || c == "ch") return 4;

//if (c == "s" || c == "c" || c == "t" || c == "l") return 4;

if (c == "^") return 3;

if (c == "\*" || c == "/") return 2;

if (c == "+" || c == "-") return 1;

if (c == "(" || c == ")") return 0;

return -1;

}

string PostfixClass::outstring()

{

return postfix;

}

void PostfixClass::ConvertToPostfix()

{

string symbol;

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

{

if (infix[i] == '0' || infix[i] == '1' || infix[i] == '2' || infix[i] == '3' || infix[i] == '4' || infix[i] == '5' || infix[i] == '6' || infix[i] == '7' || infix[i] == '8'

|| infix[i] == '9' || infix[i] == '.')

{

postfix = postfix + infix[i];

}

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

{

symbol = infix[i];

while (GetPriority(symbol) <= GetPriority(stack.getTop()))

{

postfix = postfix + " " + stack.pop\_back();

}

stack.push\_back(symbol);

postfix = postfix + " ";

}

else if ((infix.length()) - i > 2 && infix[i] == 's' && infix[i + 1] == 'i' && infix[i + 2] == 'n')// sin

{

while (GetPriority("sin") <= GetPriority(stack.getTop()))

{

postfix = postfix + stack.pop\_back();

}

stack.push\_back("sin");

postfix = postfix + " ";

i += 2;

}

else if ((infix.length())-i>2 && infix[i] == 'c' && infix[i + 1] == 'o' && infix[i + 2] == 's')//cos

{

while (GetPriority("cos") <= GetPriority(stack.getTop()))

{

postfix = postfix + " " + stack.pop\_back();

}

stack.push\_back("cos");

postfix = postfix + " ";

i += 2;

}

else if ((infix.length()) - i > 1 && infix[i] == 't' && infix[i + 1] == 'g') // tg

{

while (GetPriority("tg") <= GetPriority(stack.getTop()))

{

postfix = postfix + " " + stack.pop\_back();

}

stack.push\_back("tg");

postfix = postfix + " ";

i ++;

}

else if ((infix.length()) - i > 2 && infix[i] == 'c' && infix[i + 1] == 't' && infix[i + 2] == 'g') // ctg

{

while (GetPriority("ctg") <= GetPriority(stack.getTop()))

{

postfix = postfix + " " + stack.pop\_back();

}

stack.push\_back("ctg");

postfix = postfix + " ";

i += 2;

}

else if ((infix.length()) - i > 3 && infix[i] == 's' && infix[i + 1] == 'q' && infix[i + 2] == 'r' && infix[i + 3] == 't') // sqrt

{

while (GetPriority("sqrt") <= GetPriority(stack.getTop()))

{

postfix = postfix + " " + stack.pop\_back();

}

stack.push\_back("sqrt");

postfix = postfix + " ";

i += 3;

}

else if ((infix.length()) - i > 2 && infix[i] == 'l' && infix[i + 1] == 'o' && infix[i + 2] == 'g') // log

{

while (GetPriority("log") <= GetPriority(stack.getTop()))

{

postfix = postfix + " " + stack.pop\_back();

}

stack.push\_back("log");

postfix = postfix + " ";

i += 2;

}

else if ((infix.length()) - i > 1 && infix[i] == 'l' && infix[i + 1] == 'n') // ln

{

while (GetPriority("ln") <= GetPriority(stack.getTop()))

{

postfix = postfix + " " + stack.pop\_back();

}

stack.push\_back("ln");

postfix = postfix + " ";

i += 1;

}

else if ((infix.length()) - i > 1 && infix[i] == 'c' && infix[i + 1] == 'h') // ch

{

while (GetPriority("ch") <= GetPriority(stack.getTop()))

{

postfix = postfix + " " + stack.pop\_back();

}

stack.push\_back("ch");

postfix = postfix + " ";

i += 1;

}

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

{

stack.push\_back("(");

}

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

{

symbol = stack.pop\_back();

while (symbol != "(")

{

postfix = postfix+ " " + symbol;

symbol = stack.pop\_back();

}

}

else if ((infix.length()) - i > 1 && infix[i]=='p' && infix[i+1]=='i')//Pi

{

postfix = postfix + "pi";

i++;

}

else if (infix[i] == 'e')//E

{

postfix = postfix + "e";

}

}

while (!stack.isEmpty())

{

postfix = postfix+ " " + stack.pop\_back();

}

}

double PostfixClass::Calculate()

{

string symbol;

double op1;

double op2;

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

{

if (postfix[i] == '0' || postfix[i] == '1' || postfix[i] == '2' || postfix[i] == '3' || postfix[i] == '4' || postfix[i] == '5' || postfix[i] == '6' || postfix[i] == '7' || postfix[i] == '8'

|| postfix[i] == '9' || postfix[i] == '.')

{

string num;

while (postfix[i] == '0' || postfix[i] == '1' || postfix[i] == '2' || postfix[i] == '3' || postfix[i] == '4' || postfix[i] == '5' || postfix[i] == '6' || postfix[i] == '7' || postfix[i] == '8'

|| postfix[i] == '9' || postfix[i] == '.')

{

num += postfix[i];

i++;

}

i--;

stack.push\_back(num);

}

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

{

op1 = atof(stack.pop\_back().c\_str());

op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(op1 + op2));

}

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

{

op1 = atof(stack.pop\_back().c\_str());

op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(op2 - op1));

}

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

{

op1 = atof(stack.pop\_back().c\_str());

op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(op1 \* op2));

}

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

{

op1 = atof(stack.pop\_back().c\_str());

op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(op2 / op1));

}

else if (postfix[i] == '^')

{

op1 = atof(stack.pop\_back().c\_str());

op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(pow(op2,op1)));

}

else if ((postfix.length()) - i > 2 && postfix[i] == 's' && postfix[i+1] == 'i' && postfix[i+2] == 'n')

{

op1 = atof(stack.pop\_back().c\_str());

//op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(sin(op1)));

}

else if ((postfix.length()) - i > 2 && postfix[i] == 'c' && postfix[i+1] == 'o' && postfix[i+2] == 's')

{

op1 = atof(stack.pop\_back().c\_str());

//op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(cos(op1)));

}

else if ((postfix.length()) - i > 2 && postfix[i] == 'c' && postfix[i+1] == 't' && postfix[i+2] == 'g')

{

op1 = atof(stack.pop\_back().c\_str());

//op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(1 / tan(op1)));

}

else if ((postfix.length()) - i > 1 && postfix[i] == 't' && postfix[i+1] == 'g')

{

op1 = atof(stack.pop\_back().c\_str());

//op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(tan(op1)));

}

else if ((postfix.length()) - i > 1 && postfix[i] == 'l' && postfix[i+1] == 'n')

{

op1 = atof(stack.pop\_back().c\_str());

//op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(log(op1)));

}

else if ((postfix.length()) - i > 2 && postfix[i] == 'l' && postfix[i+1] == 'o' && postfix[i + 2] == 'g')

{

op1 = atof(stack.pop\_back().c\_str());

//op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(log10(op1)));

}

else if ((postfix.length()) - i > 3 && postfix[i] == 's' && postfix[i + 1] == 'q' && postfix[i + 2] == 'r' && postfix[i + 3] == 't')

{

op1 = atof(stack.pop\_back().c\_str());

//op2 = atof(stack.pop\_back().c\_str());

stack.push\_back(to\_string(sqrt(op1)));

}

else if ((postfix.length()) - i > 1 && postfix[i] == 'c' && postfix[i + 1] == 'h')//ch

{

op1 = atof(stack.pop\_back().c\_str());

////op2 = atof(stack.pop\_back().c\_str());

//stack.push\_back(to\_string(log10(op1)));

stack.push\_back(to\_string ( (pow(E, op1)+pow(E, -op1))/2 ) );

}

else if ((postfix.length()) - i > 1 && postfix[i] == 'p' && postfix[i + 1] == 'i' )

{

//op1 = atof(stack.pop\_back().c\_str());

////op2 = atof(stack.pop\_back().c\_str());

//stack.push\_back(to\_string(log10(op1)));

stack.push\_back(to\_string(PI));

}

else if (postfix[i] == 'e')

{

//op1 = atof(stack.pop\_back().c\_str());

////op2 = atof(stack.pop\_back().c\_str());

//stack.push\_back(to\_string(log10(op1)));

stack.push\_back(to\_string(E));

}

}

double result = atof(stack.pop\_back().c\_str());

return result;

}

Stack.cpp

#include <stdexcept>

#include "iostream"

#include "Stack.h"

using namespace std;

void Stack::reset\_list()

{

head = nullptr;

tail = nullptr;

}

Stack::Stack()

{

reset\_list();

size = 0;

}

Stack::~Stack()

{

}

void Stack::push\_back(string newElem)

{

if (size == 0) {

head = new StackNode(newElem);

tail = head;

}

else {

tail->next = new StackNode(newElem, nullptr, tail);

tail = tail->next;

}

size++;

}

string Stack::pop\_back()

{

string temp;

if (size == 0) return " ";

if (size == 1) {

temp = head->data;

delete head;

reset\_list();

}

else {

temp = tail->data;

StackNode\* current = tail->prev;

current->next = nullptr;

delete tail;

tail = current;

}

size--;

return temp;

}

string Stack::getTop()

{

if (size == 0) return " ";

return tail->data;

}

size\_t Stack::get\_size()

{

return size;

}

void Stack::clear()

{

StackNode\* current = head->next;

while (current->next != nullptr) {

delete head;

head = current;

current = head->next;

}

delete head;

delete tail;

size = 0;

}

bool Stack::isEmpty() // if empty - true, else false

{

return !size;

}