Министерство образования и науки Российской Федерации

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

Институт Информационных технологий, математики и механики

Отчёт по лабораторной работе

Вычисление арифметических выражений

Выполнил:

студент гр. 381806-1

Емельховская Е. Е.

Проверил:

ассистент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2019 г.

**Содержание**

[Введение 3](#_gjdgxs)

[Постановка задачи 4](#_30j0zll)

[Руководство пользователя 5](#_1fob9te)

[Руководство программиста 7](#_2et92p0)

[Описание структуры программы 7](#_2et92p0)

[Описание структур данных 8](#_tyjcwt)

[Описание алгоритмов 9](#_3dy6vkm)

[Заключение 14](#_2s8eyo1)

[Литература 15](#_44sinio)

[Приложения 16](#_2jxsxqh)

[Приложение 1. Исходный код основной функции 16](#_z337ya)

[Приложение 2. Класс Stack 17](#_lnxbz9)

[Приложение 3. Класс Exception 19](#_35nkun2)

[Приложение 3. Класс Postfix 20](#_1ksv4uv)

**Введение**

Математика, вычисления являются неотъемлемой частью нашей жизни. Вычисления необходимы в любом деле и в любых профессиях. Но чтобы они проходили быстро и безболезненно, можно написать программу, которая будет проводить все необходимые операции. С этой задачей прекрасно справляется Польская форма записи(постфиксная), которую я и собираюсь реализовать сегодня.

**Постановка задачи**

**Задача:** Разработать программу, преобразовывающую арифметические выражения в постфиксную форму, принимающую значения пользователя и подсчитывающую результат.

**Входные данные:**

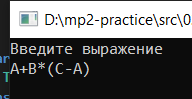
1. Строка с выражением
2. Значения переменных

**Выход:**

1. Результат подсчета

**Руководство пользователя**

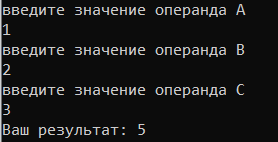
1. Запустите приложение и введите выражение, которое хотите вычислить



1. Если вы корректно ввели выражение и не забыли про скобки, то на экране вы увидите постфиксную форму записи. Далее на экране появится кол-во ваших операндов(без повторений).



1. Введите поочередно значения операндов и получите результат:)



**Руководство программиста**

***Описание структуры программы***

Список файлов:

* main.cpp (реализация главной задачи)
* calculator.h (класс TCalculator и его методы)
* exceptions.h (класс исключений)
* tstack.h (класс TStack и его методы)

***Описание структур данных***

int TCalculator:: Priority(const char sign)

**Назначение:** определяет приоритет операций

**Входные данные:** знак операции

**Выходные данные:** число, обозначающее приоритет операции

bool TCalculator::Comparison(char exp, TStack<char>& pop\_el)

**Назначение:** сравнение приоритетов знака с остальными знаками из стека операций

**Входные данные:** знак операции и стек операций

**Выходные данные:** true или false (результат сравнения)

bool TCalculator::IsItOperation(const char sign)

**Назначение:** определяет, операция это или нет

**Входные данные**: знак

**Выходные данные:** true или false

int TCalculator::GettingCount(string postfix\_form)

**Назначение:** получение кол-ва неповторяющихся операндов

**Входные данные**: строка с постфиксной формой

**Выходные данные:** кол-во

void TCalculator::GettingValues(double\*values,string&operands, string p\_f, int count)

**Назначение:** принимает значения пользователя

**Входные данные:** массив значений, строка операндов, постфиксная форма, кол-во операндов

string TCalculator::PostfixForm(string exp)

**Назначение:** получение постфиксной формы

**Входные данные:** строка с выражением пользователя

**Выходные данные**: строка с постфиксной формой

double TCalculator::Calculate(double\* values, string& operands, string p\_f)

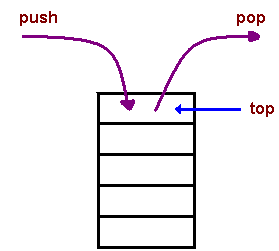
**Назначение:** подсчет результата

**Входные данные:** массив значений, строка операндов, строка с постфиксной формой

**Выходные данные:** результат

***Описание алгоритмов***

В данной программе реализована структура стек. Стек представляет собой список элементов, организованный по принципу LIFO, то есть «последним пришел, первым вышел». Количество элементов можно узнать по индексу первого свободного в стеке элементу(top).



Из чего состоит стек:

* Максимальное размер стека(сколько элементов может вместится) - max\_size
* Массив элементов - elements
* Индекс вершины стека(индекс первого свободного элемента) - top

Операции над стеком:

* Положить элемент - Push
* Элемент на вершине стека - Pop\_Get
* Удалить элемент - Pop
* Проверка на полноту - IsFull
* Проверка на пустоту - IsEmpty

**Алгоритм перевода в постфиксную форму и вычисления по ней:**

Приоритет операций:

* ‘\*’ и ‘/’ – приоритет 1
* ‘+’ и ‘—’ – приоритет 2
* остальные - приоритет 3

Преобразование выражений в польскую запись:

Выражение, введенное пользователем - *exp*. Для хранения операций будем использовать стек *stack1*, а для операций *stack2*.

1. Пройдемся по строке *exp*.
   1. Если символ является пробелом, то переходим к следующему символу.
   2. Если символ является операцией, то кладем его в *stack1*. При этом, если приоритет операции на вершине стека *stack1* выше, то перекладываем все операции из *stack1*, у которых приоритет выше, в *stack2.* И уже только после этого кладем операцию в *stack1.*
   3. Если символ является ‘*(*‘, то кладем в *stack2*.
   4. Если символ является ‘*)*’, то все операции из *stack1* переносятся в *stack2* до появления левой скобки. Левую скобку удаляем из *stack1*.
2. После прохода по *exp*, все оставшиеся операции из *stack1* переносим в *stack2*.
3. Из *stack2* берем постфиксную запись.

Вычисление значения выражения:

Создадим результирующий массив resulting\_mas. p\_f - строка с постфиксной записью. values operands - массивы для хранения значений операндов и самих операндов.

1. Просматриваем p\_f:

* 1. Если встретилась переменная, то находим ее имя в массиве operands и ее значение из values,кладем в resulting\_mas.
  2. Если встретился символ операции, то изымаем из resulting\_mas сначала первый элемент – *first*, а затем второй – *second*. Выполняем данную операцию в порядке названий переменных, то есть: *first* <*операция*> *second*. Значение операции кладем resulting\_mas.

1. По окончании просмотра строки в resulting\_mas будет хранится единственная переменная – значение выражения.

**Заключение**

В ходе лабораторной работы я разработала программу, вычисляющую арифметические выражения, с помощью постфиксной формы записи. В ходе работы были реализованы класс стека, методы получения постфиксной записи и подсчета значения по ней.

**Литература**

1. http://kvodo.ru/stack-data-structure.html
2. http://natalia.appmat.ru/c&c++/postfisso.html

**Приложения**

***Приложение 1. Исходный код основной функции***

#include <iostream>

#include <string>

#include "calculator.h"

#include "TStack.h"

#include "exceptions.h"

using namespace std;

void main() {

string expression;

string operands;

setlocale(LC\_ALL, "Rus");

try {

cout << "Введите выражение" << endl;

getline(cin, expression);

string p\_f = TCalculator::PostfixForm(expression);

cout << "Постфиксная форма равна:" << endl;

cout << p\_f << endl;

int count = TCalculator::GettingCount(p\_f);

cout << "Количество неповторяющихся операндов:" << count << endl;//(вот до этого момента все супер)

double\* values = new double[count];

TCalculator::GettingValues(values, operands, p\_f, count);

double result = TCalculator::Calculate(values, operands, p\_f);

cout << "Ваш результат: " << result << endl;

}

catch (const Exception& ex){

cout << ex.what() << endl << endl;

}

system("pause");

}

***Приложение 2. Класс TStack***

#ifndef \_TSTACK\_H\_

#define \_TSTACK\_H\_

#include "exceptions.h"

using namespace std;

template<typename ValType>

class TStack {

private:

int max\_size;

ValType\* elements;

int top;

public:

TStack(int);//конструктор

TStack(const TStack&);//конструктор копирования

~TStack();//деструктор

void Push(ValType);//кладет элемент на вершину стека

ValType Pop\_Get();//удаляем элемент и возвращаем его

bool IsEmpty()const;//проверка на пустоту

bool IsFull()const;//проверка на полноту

void Pop();//уменьшаем стек

};

template<typename ValType>//констркутор с параметром

TStack<ValType>::TStack(int \_max\_size) {

if (\_max\_size <= 0) {

throw Exception("Некорректный максимальный размер стека\n");

}

max\_size = \_max\_size;

elements = new ValType[max\_size];

top = 0;

for (int i = 0; i < max\_size; i++) {

elements[i] = 0;

}

}

template<typename ValType>//конструктор копирования

TStack<ValType>::TStack(const TStack& stack) {

max\_size = stack.max\_size;

top = stack.top;

for (int i; i < max\_size; i++) {

elements[i] = stack.elements[i];

}

}

template<typename ValType>//деструктор(done)

TStack<ValType>::~TStack() {

top = 0;

max\_size = 0;

delete [] elements;

}

template<typename ValType>//добавление элемента на вершину стека

void TStack<ValType>::Push(ValType el) {

if (IsFull()) {

throw Exception("Стек полон\n");

}

elements[top] = el;

top++;

}

template<typename ValType>//удаление элемента и получение топа

ValType TStack<ValType>::Pop\_Get() {

if (IsEmpty()) {

throw Exception("Стек пуст\n");

}

return elements[top-1];

}

template<typename ValType>//проверка на пустоту

bool TStack<ValType>::IsEmpty()const{

return (top == 0);

}

template<typename ValType>//проверка на полноту

bool TStack<ValType>::IsFull()const {

return (max\_size == top);

}

template<typename ValType>//уменьшение стека

void TStack<ValType>::Pop() {

if (IsEmpty()) {

throw Exception("Стек пуст\n");

}

top--;

}

#endif

***Приложение 3. Класс Exception***

#ifndef \_EXCEPTIONS\_H\_

#define \_EXCEPTIONS\_H\_

#include <iostream>

#include <string>

#include <exception>

using namespace std;

class Exception : public exception{

private:

string msg;

public:

Exception(string \_msg) : msg(\_msg) {};

const char\* what() const noexcept{

return msg.c\_str();

}

};

#endif

***Приложение 3. Класс TCalculator***

#include <iostream>

#include <string>

#include "tstack.h"

#include "exceptions.h"

using namespace std;

class TCalculator {

private:

static int Priority(const char);//определение приоретета операций

static bool Comparison(char, TStack<char>&);//сравнение приоритетов

static bool IsItOperation(const char);//определение операция это или нет

public:

static string PostfixForm(string);//образование постфиксной формы

static int GettingCount(string);//возвращает кол-во операндов

static double Calculate(double\*, string&, string);//возвращает результат подсчета

static void GettingValues(double\*, string&, string, int);//ввод значений

};

//определение приоретета операций

int TCalculator:: Priority(const char sign) {

switch (sign) {

case '+': return 2;

case '-': return 2;

case '\*': return 1;

case '/': return 1;

default: return 3;

}

}

bool TCalculator::Comparison(char exp, TStack<char>& pop\_el){//сравнение приоритетов

return (Priority(pop\_el.Pop\_Get()) < Priority(exp));

};

//определение операция это или нет

bool TCalculator::IsItOperation(const char sign) {

return ((sign == '\*') || (sign == '/') || (sign == '+') || (sign == '-'));

}

//возвращает кол-во неповторяющих операндов

int TCalculator::GettingCount(string postfix\_form) {

int count = 0;

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

if (isalpha(postfix\_form[i])) {

count++;

}

}

return count;

}

//ввод значений

void TCalculator::GettingValues(double\* values, string& operands, string p\_f, int count) {

int current\_count\_of\_operands = 0;

double value = 0;

char\* new\_operands = new char[count];

double\* new\_values = new double[count];

for (int i = 0; i < p\_f.length(); i++) {

if (isalpha(p\_f[i])) {

int flag = 0;

for (int j = 0; j < current\_count\_of\_operands; j++) {

if (new\_operands[j] == p\_f[i]) {

flag = 1;

break;

}

}

if (flag == 0) {

new\_operands[current\_count\_of\_operands] = p\_f[i];

cout << "введите значение операнда " << p\_f[i] << endl;

cin >> value;

new\_values[current\_count\_of\_operands] = value;

current\_count\_of\_operands++;

}

}

}

operands.assign(new\_operands);//скопировали 2ое в 1ое, преобразовали к string

memcpy(values, new\_values, sizeof(double) \* count);//скопировали 2ое в 1ое

}

//образование постфиксной формы

string TCalculator::PostfixForm(string exp) {

if (exp.length() == 0) {

throw Exception("Некорректно введенная строка\n");

}

TStack<char> stack1(exp.length() + 1);//операции

TStack<char> stack2(exp.length() + 1);//операнды

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

char sign = static\_cast<char>(exp[i]);

if (sign == ' ') {

continue;

}

if (IsItOperation(sign)) {

if (stack1.IsEmpty()) {//если в стеке опраций еще ничего нет

stack1.Push(sign);

continue;

}

if (Comparison(sign, stack1)) {//битва за приоритет

while (!stack1.IsEmpty()) {

stack2.Push(stack1.Pop\_Get());

stack1.Pop();

}

stack1.Push(sign);

}

else

stack1.Push(sign);

}

if (sign == '(') {

stack1.Push(sign);

}

if (isalpha(sign)) {

stack2.Push(sign);

}

if (sign == ')') {

int left\_bracket\_flag = 0;

while (!stack1.IsEmpty()) {

if (stack1.Pop\_Get() != '(') {

cout << stack1.Pop\_Get() << endl;

stack2.Push(stack1.Pop\_Get());

stack1.Pop();

continue;

}

stack1.Pop();

left\_bracket\_flag = 1;

break;

}

if ((left\_bracket\_flag != 1) && (stack1.IsEmpty())) {

throw Exception(" Кажется, вы забыли скобку (\n");

}

}

}

while (!stack1.IsEmpty()){

stack2.Push(stack1.Pop\_Get());

stack1.Pop();

}

string postfix\_form;

while (!stack2.IsEmpty()){

postfix\_form += stack2.Pop\_Get();

stack2.Pop();

}

for (int i = 0; i < postfix\_form.length() / 2; i++)

swap(postfix\_form[i], postfix\_form[postfix\_form.length() - 1 - i]);

return postfix\_form;

}

//подсчет

double TCalculator::Calculate(double\* values, string& operands, string p\_f)

{

TStack<double> resulting\_mas(p\_f.length());

for (int i = 0; i < p\_f.length(); i++){

char sign = static\_cast<char>(p\_f[i]);

if (isalpha(sign)){

for (int j = 0; j < operands.length(); j++){

if (operands[j] == sign){

resulting\_mas.Push(values[j]);

break;

}

}

continue;

}

double first = resulting\_mas.Pop\_Get();

resulting\_mas.Pop();

double second = resulting\_mas.Pop\_Get();

resulting\_mas.Pop();

double result;

switch (sign){

case '+':

result = second + first;

break;

case '-':

result = second - first;

break;

case '\*':

result = second \* first;

break;

case '/':

if (first == 0)

throw Exception("На 0 делить нельзя:(\n");

result = second / first;

break;

}

resulting\_mas.Push(result);

}

return resulting\_mas.Pop\_Get();

};