Министерство науки и высшего образования РФ

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

Институт информационных технологий, математики и механики

Отчет по учебной практике

**Вычисление арифметических выражений**

Выполнил: Параничева Алена Владиславовна

студент группы 381806-1

Проверил: к. т. н., доцент кафедры МОСТ

Кустикова Валентина Дмитриевна

Нижний Новгород

2019

**Содержание**

[Введение 3](#_Toc28007563)

[Постановка задачи 4](#_Toc28007564)

[1. Руководство пользователя 5](#_Toc28007565)

[2. Руководство программиста 7](#_Toc28007566)

[2.1. Структура программы 7](#_Toc28007567)

[2.2. Описание структур данных 7](#_Toc28007568)

[2.2.1. Класс TStack 7](#_Toc28007569)

[2.2.2. Класс Calculator 9](#_Toc28007570)

[2.3. Описание алгоритмов 10](#_Toc28007571)

[2.3.1. Представление стека 10](#_Toc28007572)

[2.3.2. Алгоритм перевода арифметического выражения в постфиксную форму 11](#_Toc28007573)

[2.3.3. Алгоритм вычисления значения выражения по постфиксной форме 12](#_Toc28007574)

[Заключение 14](#_Toc28007575)

[Литература 15](#_Toc28007576)

[Приложение 16](#_Toc28007577)

# Введение

В обычной жизни нам часто приходится сталкиваться с большими примерами, которые часто имеют множество действий, состоящие из различных операций и скобок. Очевидно, человеку не так-то просто самостоятельно посчитать такого рода примеры, при этом не сделав ни единой ошибки. Однако, с помощью калькулятора со скобками, можно быстро и точно посчитать выражение любой длины.

Чтобы реализовать такой калькулятор необходим стек. Стек – структура данных представляющая собой список элементов, организованных по принципу «последним пришёл – первым вышел». Концептуально, структура данных – стек очень проста: она позволяет добавлять или удалять элементы в определенном порядке. Каждый раз, когда добавляется элемент, он попадает на вершину стека, единственный элемент, который может быть удален из стека – элемент, который находится на вершине стека.  Таким образом, сама структура данных обеспечивает надлежащий порядок вызовов. Последняя добавленная в стек функция, будет завершена первой и наоборот, первая добавленная в стек функция будет завершена последней.

Также чтобы реализовать калькулятор, необходима определенная запись выражения – запись, в которой мы привыкли представлять примеры со скобками, не подходит. Чтобы программа правильно сосчитала исходный пример, выражение нужно преобразовать с помощью обратной польской записи. Обратная польская нотация – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Преимущества этой записи в том, что выражение может быть записано без скобок.

Поэтому в программе будет представлены класс Стек, обратная перезапись вводимого пользователем выражения и, конечно, его вычисление.

# Постановка задачи

Постановка задачи:

Разработать программу, преобразовывающую арифметические выражения, состоящее из односимвольных идентификаторов, в постфиксную форму и далее вычисляющую их.

Входные данные:

* Строка с арифметическим выражением
* Значения переменных

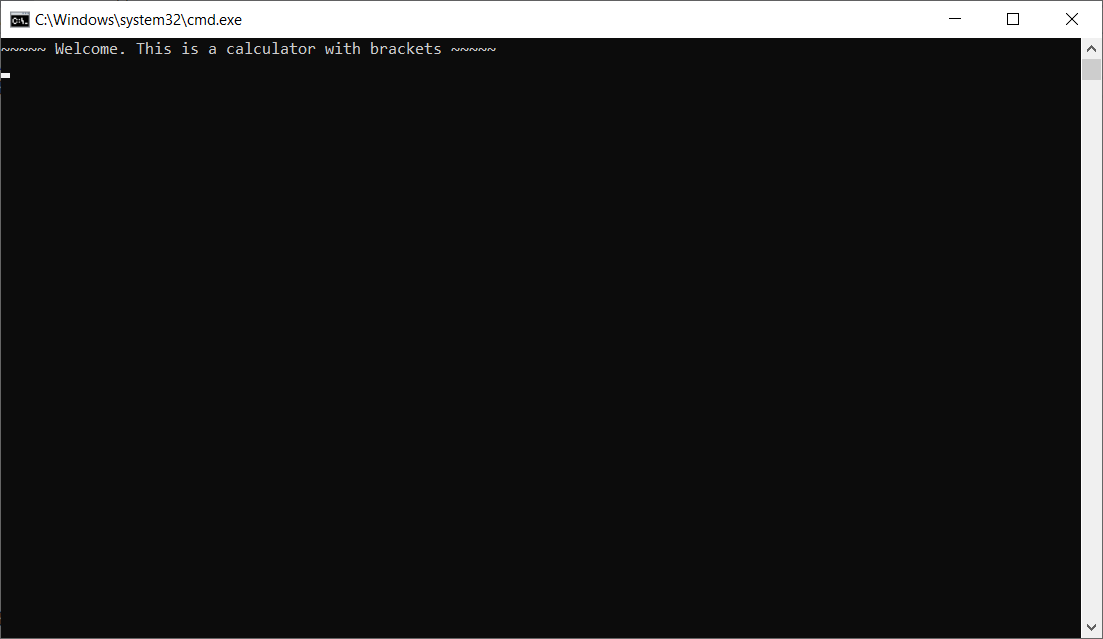
Выход:

* Результат вычисления

# Руководство пользователя

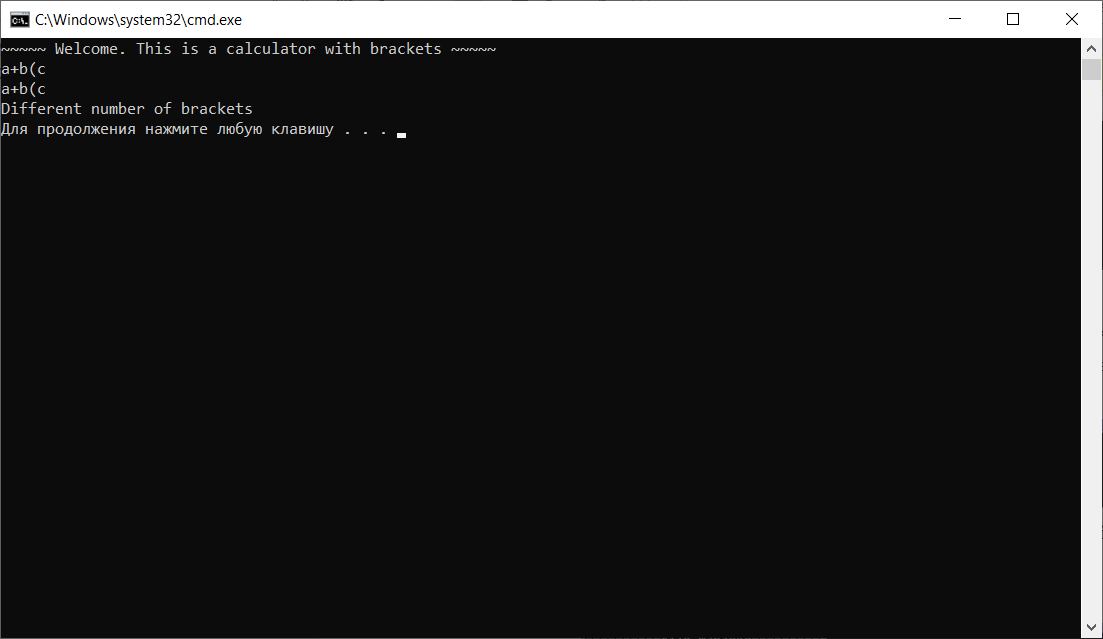
***В данном руководстве содержатся пошаговые инструкции по работе с программой:***

Запустите программу. Перед вами отобразится экран, на котором выведено приветствие (рис. 1).



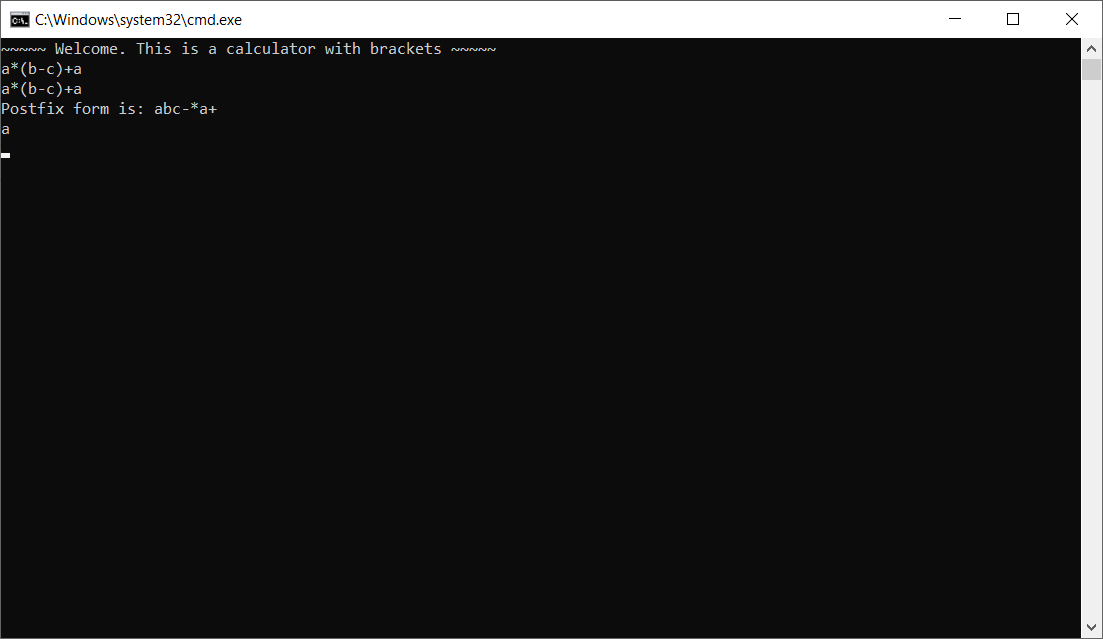
1. Программа после запуска.

С помощью клавиатуры введите выражение, которое содержит только буквы, скобки и операции сложения, вычитания, умножения и деления. Нажмите Enter. Если вы ввели выражение некорректно, программа отобразит сообщение об ошибке (рис. 2).



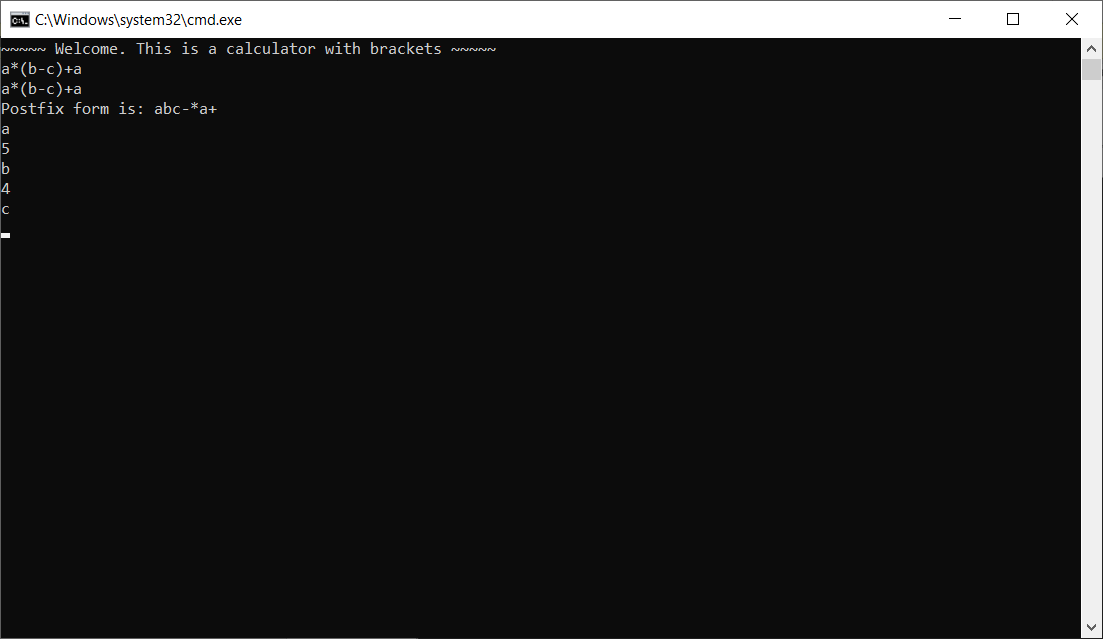
1. Сообщение об ошибке при вводе выражения с некорректной записью.

При правильном вводе выражения программа обработает его и выведет его постфиксную запись (рис. 3).



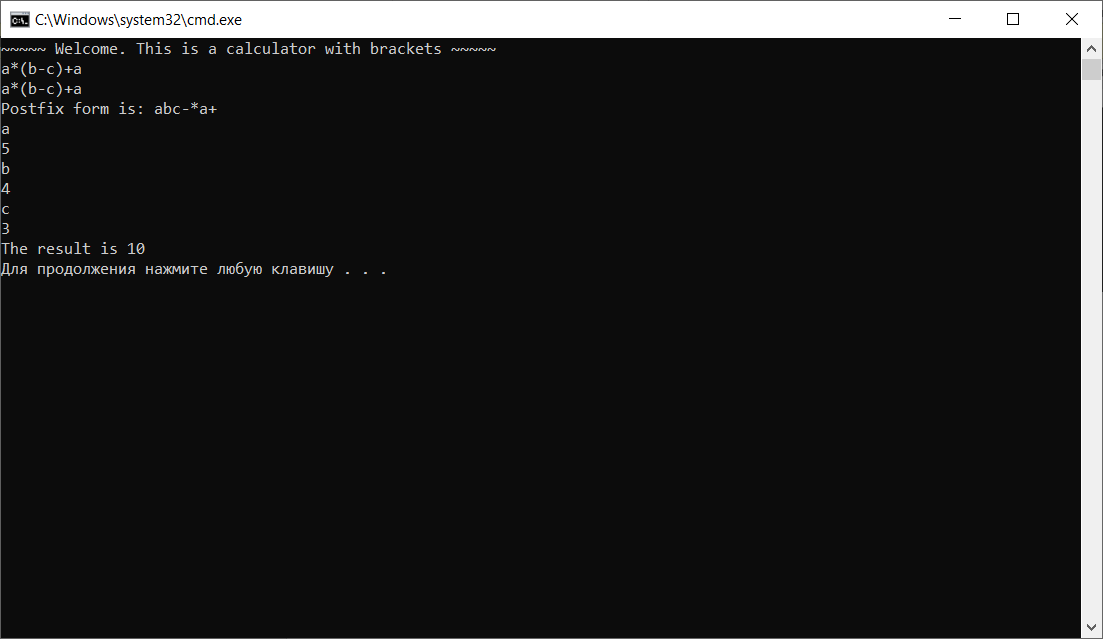
1. Пример вывода постфиксной записи.

Далее программа запросит числовой ввод операндов. После каждого ввода операнда нажмите Enter (рис. 4).



1. Пример ввода операндов.

На данном шаге программа выведет результат выражения (рис. 5).



1. Пример результата выражения.

Этот шаг является завершающим для программы.

# Руководство программиста

## Структура программы

Код программы находится в файлах:

**Main2.cpp** содержит основную функцию,

**Calculator.cpp** содержит реализация функций класса Calculator,

**HCalculator.h** содержит объявление класса Calculator,

**TStack.h** содержит объявление и реализацию шаблонного класса TStack.

## Описание структур данных

### Класс TStack

Ниже приведено объявление шаблонного класса Стек:

template<typename ValueType>

class TStack {

private:

int maxsize;

ValueType\* x;

int top;

public:

TStack(int);

TStack(const TStack&);

~TStack();

void Push(ValueType);

void Pop();

bool IsEmpty()const;

bool IsFull()const;

ValueType Top()const;

};

Поля класса:

int maxsize - максимальное количество элементов, хранящихся в стеке

ValueType\* x – тип массива элементов хранящихся в стэке

int top – Позиция первого свободного элемента массива (вершина стека)

TStack(int)

***Назначение:*** создание объекта класса TStack с заданным максимальным размером.

***Входные параметры:*** параметр типа int.

TStack(const TStack&);

***Назначение:*** создание объекта класса TStack при помощи копирования другого объекта этого класса

***Входные параметры:*** параметр типа const TStack&.

~TStack()

***Назначение:*** удаление объекта класса TStack.

***Входные параметры:*** отсутствуют.

void Push(ValueType)

***Назначение:*** вставка нового элемента в конец стэка

***Входные параметры:*** ValueType – вставляемая переменная.

***Выходные параметры:*** отсутствуют.

void Pop()

***Назначение:*** удаление верхнего элемента из стэка

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** отсутствуют.

bool IsEmpty()const

***Назначение:*** проверка на пустоту.

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** True – если стек пуст, False – если стек непустой.

bool IsFull()const

***Назначение:*** проверка на полноту стэка.

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** True – стэк заполнен полностью, False –стэк не заполнен полностью.

ValueType Top()const

***Назначение:*** возвращение верхнего элемента стэка.

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** верхний элемент стэка.

### Класс Calculator

#ifndef CALCULATOR\_H

#define CALCULATOR\_H

#include "TStack.h"

#include <string>

class Calculator

{

static int Priority(char);

public:

static bool Mistakes(std::string);

static std::string Postfix(std::string);

static void GettingValues(std::string, char\*&, double\*&, int&);

static double Calculate(std::string str, char\* let, double\* val, int n);

};

#endif

int Priority(char o)

***Назначение:*** функция определяет приоритет операций.

***Входные параметры:* о** – переменная, содержащая символ знака операции.

***Выходные параметры:*** число, определяющее приоритет операции.

bool Mistakes(std::string s)

***Назначение:*** обработка входной строки от пользователя на наличие ошибок в синтаксисе.

***Входные параметры:*** s - строка, введенная пользователем.

***Выходные параметры:*** true или false, true, если строка не имеет ошибок в синтаксисе, false, если строка имеет ошибки в синтаксисе.

std::string Postfix(std::string s)

***Назначение:*** перевод введенной строки(выражения пользователя) в постфиксную форму и ее вывод.

***Входные параметры:*** s – строка, введенная пользователем.

***Выходные параметры:*** строка в постфиксной форме.

void GettingValues(std::string str, char\*& let, double\*& val, int& n)

***Назначение:*** выделение памяти под массивы содержащие операнды и значения операндов, запрос у пользователя значений операндов.

***Входные параметры:***

str – постфиксная запись строки,

let – массив, содержащий операнды,

val – массив, содержащий значения операндов,

n – количество операндов.

***Выходные параметры:*** отсутствуют.

double Calculate(std::str, char\* let, double\* val, int n)

***Назначение:*** подсчет значения введенного пользователем выражения.

***Входные параметры:***

str – строка в постфиксной форме,

let – массив операндов,

val – массив значений операндов,

n – количество операндов.

***Выходные параметры:*** отсутствуют.

## Описание алгоритмов

### Представление стека

В программе используется такая структура данных, как стек. Стек – структура данных типа список, работающая по принципу LIFO (Последним пришел, первым вышел). Для запоминания количества хранимых в стеке значений используется индекс первого свободного элемента в векторе, в программе реализован на основе массивов.

**Характеристиками стека являются:**

1. Максимальное количество элементов, хранящихся в стеке
2. Массив элементов, хранящихся в стеке
3. Позиция первого свободного элемента массива (вершина стека)

**Операции** **над стеком:**

1. Вставка: вставка элемента на вершину стека
2. Удаление: удаление элемента стека на его вершине
3. Просмотр элемента: просмотр элемента на вершине стека
4. Проверка на пустоту: проверка пуст ли стек
5. Проверка на полноту: проверка заполнен ли полностью стек

### Алгоритм перевода арифметического выражения в постфиксную форму

Выражение перезаписывается в обратную польскую запись. Обратная польская нотация – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций.[2]

Алгоритм перевода в обратную польскую запись с использованием стека:

Нам понадобится два стека для переменных типа char, т.к. исходное выражение мы получаем в виде строки и стек для постфиксной записи.

1. Создаем два стека.
2. Берем исходную строку и рассматриваем поочередно каждый символ:
   1. Если этот символ - число (или переменная), то просто помещаем его во второй стек.
   2. Если символ - знак операции (+, -, \*, / ), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет (допустим он равен 3). Операции сложения и вычитания имеют меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.
   3. Получив один из этих символов, мы должны проверить первый стек:
      1. Если первый 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 первый стек.
      2. Если символ, находящийся на вершине первого стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из первого стека во второй стек до тех пор, пока выполняется это условие; затем переходим к пункту i).
   4. Если текущий символ - открывающая скобка, то помещаем ее в первый стек.
   5. Если текущий символ - закрывающая скобка, то извлекаем символы из первого стека во второй стек до тех пор, пока не встретим в первом стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.
   6. Если вся входная строка разобрана, а в первом стеке еще остаются знаки операций, извлекаем их из стека во второй стек.[1]

Рассмотрим алгоритм на примере простейшего выражения:  
Дано выражение:

**a + (b - c) \* d**

Рассмотрим поочередно все символы:

|  |  |  |  |
| --- | --- | --- | --- |
| **Символ** | **Действие** | **Состояние выходной строки после совершенного действия** | **Состояние стека после совершенного действия** |
| a | 'a' - переменная. Помещаем ее во второй стек | a | пуст |
| + | '+' - знак операции. Помещаем его в первый стек (поскольку первый стек пуст, приоритеты можно не проверять) | a | + |
| ( | '(' - открывающая скобка. Помещаем в первый стек. | a | + ( |
| b | 'b' - переменная. Помещаем ее во второй стек | a b | + ( |
| - | '-' - знак операции, который имеет приоритет 2. Проверяем первый стек: на вершине находится символ '(', приоритет которого равен 1. Следовательно мы должны просто поместить текущий символ '-' в первый стек. | a b | + ( - |
| c | 'c' - переменная. Помещаем ее во второй стек | a b c | + ( - |
| ) | ')' - закрывающая скобка. Извлекаем из первого стека в выходную строку все символы, пока не встретим открывающую скобку. Затем уничтожаем обе скобки. | a b c - | + |
| \* | '\*' - знак операции, который имеет приоритет 3. Проверяем стек: на вершине находится символ '+', приоритет которого равен 2, т.е. меньший, чем приоритет текущего символа '\*'. Следовательно мы должны просто поместить текущий символ '\*' в первый стек. | a b c - | + \* |
| d | 'd' - переменная. Помещаем ее во второй стек | a b c - d | + \* |

Теперь вся входная строка разобрана, но в стеке еще остаются знаки операций, которые мы должны просто извлечь в выходную строку. Поскольку стек - это структура, организованная по принципу LIFO, сначала извлекается символ '\*', затем символ '+'.

Итак, мы получили конечный результат: **a b c - d \* +**.[3]

### Алгоритм вычисления значения выражения по постфиксной форме

В качестве входной строки мы теперь рассматриваем выражение, записанное в ОПЗ.

1. Для реализации этого алгоритма используется стек для чисел (или для переменных, если они встречаются в исходном выражении).
2. Просматриваем строку слева направо.
   1. Если очередной символ входной строки - число, то кладем его в стек.
   2. Если очередной символ - знак операции, то извлекаем из стека два верхних числа, используем их в качестве операндов для этой операции, затем кладем результат обратно в стек.
   3. Когда вся входная строка будет разобрана в стеке должно остаться одно число, которое и будет результатом данного выражения.

Рассмотрим этот алгоритм на примере выражения: 7 5 2 - 4 \* +

Рассмотрим поочередно все символы:

|  |  |  |
| --- | --- | --- |
| Символ | Действие | Состояние стека после совершенного действия |
| 7 | '7' - число. Помещаем его в стек. | 7 |
| 5 | '5' - число. Помещаем его в стек. | 7 5 |
| 2 | '2' - число. Помещаем его в стек. | 7 5 2 |
| - | '-' - знак операции. Извлекаем из стека 2 верхних числа ( 5 и 2 ) и совершаем операцию 5 - 2 = 3, результат которой помещаем в стек | 7 3 |
| 4 | '4' - число. Помещаем его в стек. | 7 3 4 |
| \* | '\*' - знак операции. Извлекаем из стека 2 верхних числа ( 3 и 4 ) и совершаем операцию 3 \* 4 = 12, результат которой помещаем в стек | 7 12 |
| + | '+' - знак операции. Извлекаем из стека 2 верхних числа ( 7 и 12 ) и совершаем операцию 7 + 12 = 19, результат которой помещаем в стек | 19 |

Теперь строка разобрана и в стеке находится одно число 19, которое является результатом исходного выражения.[4]

# Заключение

В ходе практической работы была разработана и реализована программа для вычисления значений арифметических выражений с помощью постфиксной формы записи. В ходе работы реализована структура данных стек, алгоритм построения постфиксной записи, вычисление значения арифметического выражения на основании постфиксной формы.

# Литература

1. *Т. Пратт, М. Зелковиц.* Языки программирования: разработка и реализация Terrence W. Pratt, Marvin V. Zelkowitz. Programming Languages: Design and Implementation. — 4-е издание. — Питер, 2002. — 688 с. — (Классика Computer Science). — 4000 экз.
2. Дмитрий Пискарев. Алгоритмы и методы: Обратная польская запись. 21.10.2006 -[http://www.interface.ru/home.asp?artId=1492]
3. Егор Григорьев. Структура данных: стеки. 21.12.2013 -

[http://cppstudio.com/post/5155/]

Ivan Kudryavsky. Реализация класса для работы со стеком в С++. 21.01.2014

[https://master.virmandy.net/realizatsiya-klassa-dlya-rabotyi-so-stekom-v-s]

# Приложение

**main2.cpp**

#include <iostream>

#include <string>

#include "TStack.h"

#include "HCalculator.h"

using namespace std;

void main()

{

std::string s, ss;

double res;

cout << "~~~~~ Welcome. This is a calculator with brackets ~~~~~" <<'\n';

getline(cin, s);

cout << s << '\n';

try

{

if (Calculator::Mistakes(s) == true)

ss = Calculator::Postfix(s);

char\* let;

double\* val;

int n;

Calculator::GettingValues(ss, let, val, n);

res = Calculator::Calculate(ss, let, val, n);

cout << "The result is "<< res << '\n';

delete[] let;

delete[] val;

}

catch(const char\* sms)

{

cout << sms << endl;

}

}

**Calculator.cpp**

#include <iostream>

#include <string>

#include "TStack.h"

#include "HCalculator.h"

using namespace std;

int Calculator::Priority(char o)

{

switch (o)

{

case '\*':

return 3;

case '/':

return 3;

case '+':

return 2;

case '-':

return 2;

case '(':

return 1;

default:

cout << "Wrong Operation!" << o;

break;

}

}

bool Calculator::Mistakes(std::string s)

{

if (s.length() == 0)

throw "The line is empty";

if (s[0] == ')')

throw "You cannot start with a bracket )";

int r = 0, l = 0, a = 0, o = 0;

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

{

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

{

if (i != 0)

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

throw "Brackets together )(";

r++;

}

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

{

if (i != 0)

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

throw "Brackets together ()";

l++;

}

if (isalpha(s[i]))

a++;

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

o++;

}

if ((r > l) || (l > r))

throw "Different number of brackets";

if (o >= a)

throw "Not enough operands";

if ((o + 1) != a)

throw "Too mach operands";

return true;

}

std::string Calculator::Postfix(std::string s)

{

int n = 0;

TStack<char> post(20);

TStack<char> stack(20);

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

{

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

stack.Push(s[i]);

if (isalpha(s[i]))

{

post.Push(s[i]);

}

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

{

if (!stack.IsEmpty())

if (Priority(stack.Top()) >= Priority(s[i]))

{

post.Push(stack.Top());

stack.Pop();

}

stack.Push(s[i]);

}

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

{

while (stack.Top() != '(')

{

post.Push(stack.Top());

stack.Pop();

}

stack.Pop();

}

}

while (!stack.IsEmpty())

{

post.Push(stack.Top());

stack.Pop();

}

char tmp[20];

int i = 0;

while (!post.IsEmpty())

{

tmp[i] = post.Top();

post.Pop();

i++;

}

for (int j = i; j < 20; j++)

tmp[j] = '\0';

std::string str = string(tmp);

for (int j = 0; j < i / 2; j++)

swap(str[j], str[i - j - 1]);

cout << "Postfix form is: " << str << '\n';

return str;

}

void Calculator::GettingValues(std::string str, char\*& let, double\*& val, int& n)

{

n = 0;

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

if (isalpha(str[i]))

n++;

int f;

let = new char[n];

val = new double[n];

n = 0;

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

{

if (isalpha(str[i]))

{

f = 0;

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

if (str[i] == let[j])

f++;

if (f == 0)

{

let[n] = str[i];

n++;

}

}

}

for (int i = 0; i < n; i++)

{

cout << let[i] << '\n';

cin >> val[i];

}

}

double Calculator::Calculate(std::string str, char\* let, double\* val, int n)

{

TStack<double> stack(str.length());

char tmp;

double a = 0, b = 0, sum = 0;

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

{

tmp = str[i];

if (isalpha(tmp))

{

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

if (tmp == let[j])

{

stack.Push(val[j]);

break;

}

}

if (tmp == '\*')

{

b = stack.Top();

stack.Pop();

a = stack.Top();

stack.Pop();

sum = a \* b;

stack.Push(sum);

}

if (tmp == '+')

{

b = stack.Top();

stack.Pop();

a = stack.Top();

stack.Pop();

sum = a + b;

stack.Push(sum);

}

if (tmp == '-')

{

b = stack.Top();

stack.Pop();

a = stack.Top();

stack.Pop();

sum = a - b;

stack.Push(sum);

}

if (tmp == '/')

{

b = stack.Top();

stack.Pop();

a = stack.Top();

stack.Pop();

sum = a / b;

stack.Push(sum);

}

}

return sum;

}

HCalculator.h

#ifndef CALCULATOR\_H

#define CALCULATOR\_H

#include "TStack.h"

#include <string>

class Calculator

{

static int Priority(char);

public:

static bool Mistakes(std::string);

static std::string Postfix(std::string);

static void GettingValues(std::string, char\*&, double\*&, int&);

static double Calculate(std::string str, char\* let, double\* val, int n);

};

#endif

TStack.h

#ifndef TSTACK\_H

#define TSTACK\_H

#include <string>

template<typename ValueType>

class TStack {

private:

int maxsize;

ValueType\* x;

int top;

public:

TStack(int);

TStack(const TStack&);

~TStack();

void Push(ValueType);

void Pop();

bool IsEmpty()const;

bool IsFull()const;

ValueType Top()const;

};

/////////////////////////////////////////

template<typename ValueType>

TStack<ValueType>::TStack(int \_maxsize)

{

if (\_maxsize <= 0)

throw "Incorrect max size";

maxsize = \_maxsize;

x = new ValueType[maxsize];

top = 0;

for (int i = 0; i < maxsize; i++)

x[i] = 0;

}

template<typename ValueType>

TStack<ValueType>::TStack(const TStack& s)

{

maxsize = s.maxsize;

top = s.top;

for (int i; i < maxsize; i++)

x[i] = s.x[i];

}

template<typename ValueType>

TStack<ValueType>::~TStack()

{

top = 0;

maxsize = 0;

delete[] x;

}

template<typename ValueType>

void TStack<ValueType>::Push(ValueType a) {

if (IsFull())

{

throw "The stack is full";

}

x[top++] = a;

}

template<typename ValueType>

void TStack<ValueType>::Pop() {

if (IsEmpty())

throw "The stack is empty";

--top;

}

template<typename ValueType>

bool TStack<ValueType>::IsEmpty() const

{

return (top == 0);

}

template<typename ValType>

bool TStack<ValType>::IsFull() const

{

return (maxsize == top);

}

template<class ValueType>

ValueType TStack<ValueType>::Top()const

{

return x[top - 1];

}

#endif