Министерство образования и науки РФ

Нижегородский государственный университет имени Н.И. Лобачевского (Национальный исследовательский университет)

Институт Информационных Технологий Математики и Механики  
Отчёт по лабораторной работе №3

Тема: «Вычисление арифметических операций»



Выполнил:

студент группы 0823-3

Латанов Владислав Александрович

Проверил: Козинов Е.А.

г. Нижний Новгород 2016 год.

**Содержание.**

[Введение 3](#_Toc437886815)

[Постановка учебно-практической задачи 4](#_Toc437886816)

[Руководство пользователя 5](#_Toc437886817)

[Руководство программиста 7](#_Toc437886818)

[Заключение 9](#_Toc437886819)

[Список используемой литературы: 10](#_Toc437886820)

[Приложение 11](#_Toc437886821)

## Введение.

Тема моей лабораторной работы – вычисления арифметических выражения и вывод их в ОПЗ (обратной польской записи) на основе стека. Начнем с понятия стека. **Стек** — абстрактный тип данных, представляющий собой список элементов, организованных по «последним пришёл — первым вышел».

Моя программа должна удовлетворять пользователя и производить правильное вычисление над всеми операциями, которые перед ней поставлены. С помощью моей программы, пользователь сможет быстро и качественно производить необходимые вычисления и быть уверенным в их правильности. В программе реализованы такие операции, как: +, -, \*, /. Так же стоит отметить, что в программе реализован алгоритм расставления скобок, тем самым пользователь не допустит ошибки, при закрытии скобок.  
 Выше я упомянул об словосочетании ОПЗ (обратная польская запись). **Обратная польская записи (ОПЗ)** — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Такая запись близка к компьютеру и позволяет ему производить вычисления более быстро.

# 

## Постановка учебно-практической задачи.

Данный разрабатываемый программный комплекс должен обеспечивать:

* Ввод арифметического выражения, которое необходимо вычислить;
* Правильность расстановки приоритетов операций;
* Контроль расстановки скобок;
* Перевод арифметического выражения в ОПЗ (обратную польскую запись);
* Вычисление арифметического выражения в ОПЗ ( обратной польской записи);
* Вывод результата вычислений.

## Руководство пользователя.

Программа предлагает набор возможностей:

* Ввод математической формулы;
* Реализация операций: +, -, \*, /.
* Проверка на корректность выражения, а именно: Правильность расстановки скобок и не повторения операций: ++,--,\*\*,//,+-,+\* и т.д.
* Поверка приоритетов операций.
* Вывод результата.

## Руководство программиста.

Отличительной особенностью обратной польской нотации является то, что все аргументы (или операнды) расположены перед знаком операции. В общем виде запись выглядит следующим образом:

Запись набора операций состоит из последовательности операндов и знаков операций. Операнды в выражении при письменной записи разделяются пробелами.

Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.

Результатом вычисления выражения становится результат последней вычисленной операции.

Например, рассмотрим вычисление выражения 7 2 3 \* - (эквивалентное выражение в инфиксной нотации: 7-2\*3).

Первый по порядку знак операции — «\*», поэтому первой выполняется операция умножения над операндами 2 и 3 (они стоят последними перед знаком). Выражение при этом преобразуется к виду 7 6 - (результат умножения — 6, — заменяет тройку «2 3 \*»).

Второй знак операции — «-». Выполняется операция вычитания над операндами 7 и 6.

Вычисление закончено. Результат последней операции равен 1, это и есть результат вычисления выражения.

Очевидное расширение обратной польской записи на унарные, тернарные и операции с любым другим количеством операндов: при использовании знаков таких операций в вычислении выражения операция применяется к соответствующему числу последних встретившихся операндов.

Особенности обратной польской записи следующие:

Порядок выполнения операций однозначно задаётся порядком следования знаков операций в выражении, поэтому отпадает необходимость использования скобок и введения приоритетов и ассоциативности операций.

В отличие от инфиксной записи, невозможно использовать одни и те же знаки для записи унарных и бинарных операций. Так, в инфиксной записи выражение 5 \* (-3 + 8) использует знак «минус» как символ унарной операции (изменение знака числа), а выражение (10 - 15) \* 3 применяет этот же знак для обозначения бинарной операции (вычитание). Конкретная операция определяется тем, в какой позиции находится знак. Обратная польская запись не позволяет этого: запись 5 3 - 8 + \* ( условный аналог первого выражения) будет интерпретирована как ошибочная, поскольку невозможно определить, что «минус» после 5 и 3 обозначает не вычитание; в результате будет сделана попытка вычислить сначала 5 - 3, затем 2 + 8, после чего выяснится, что для операции умножения не хватает операндов. Чтобы всё же записать это выражение, придётся либо переформулировать его, либо ввести для операции изменения знака отдельное обозначение, например, «±»: 5 3 ± 8 + \*.

Так же, как и в инфиксной нотации, в ОПН одно и то же вычисление может быть записано в нескольких разных вариантах. Например, выражение(10 - 15) \* 3 в ОПН можно записать как 10 15 - 3 \*, а можно — как 3 10 15 - \*

Из-за отсутствия скобок обратная польская запись короче инфиксной. За этот счёт при вычислениях на калькуляторах повышается скорость работы оператора (уменьшается количество нажимаемых клавиш), а в программируемых устройствах сокращается объём тех частей программы, которые описывают вычисления. Последнее может быть немаловажно для портативных и встроенных вычислительных устройств, имеющих жёсткие ограничения на объём памяти.

**Описание алгоритмов:**

Пока есть ещё символы для чтения:

Читаем очередной символ.

Если символ является числом, добавляем его к выходной строке.

Если символ является символом функции, помещаем его в стек.

Если символ является открывающей скобкой, помещаем его в стек.

Если символ является закрывающей скобкой:

До тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется. Если после этого шага на вершине стека оказывается символ функции, выталкиваем его в выходную строку. Если стек закончился раньше, чем мы встретили открывающую скобку, это означает, что в выражении либо неверно поставлен разделитель, либо не согласованы скобки.

Если символ является оператором ***о1***, тогда:

пока…

… (если оператор ***o1*** право-ассоциированный) приоритет ***o1*** меньше приоритета оператора, находящегося на вершине стека…

… (если оператор ***o1*** ассоциированный, либо лево-ассоциированный) приоритет ***o1*** меньше либо равен приоритету оператора, находящегося на вершине стека…

… выталкиваем верхние элементы стека в выходную строку;

помещаем оператор ***o1*** в стек.

Когда входная строка закончилась, выталкиваем все символы из стека в выходную строку. В стеке должны были остаться только символы операторов; если это не так, значит в выражении не согласованы скобки.

**Структура программного комплекса:**

*TStack.h -* заголовочный файл класса TStack.

*TStack.cpp -* файл реализации класса TStack.  
Prot.h – файл, где объявлены протатипы функций.

*Alg.cpp –* файл, с описание функций.

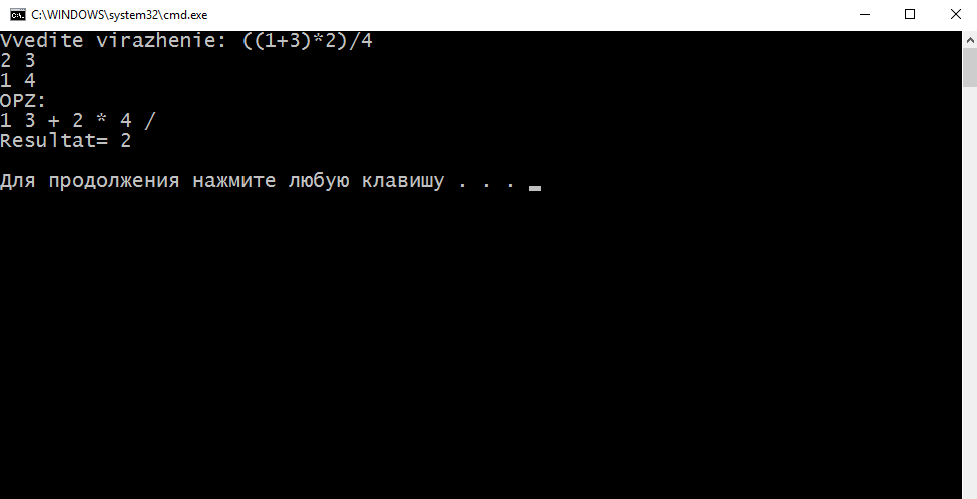
*main.cpp –* главная функция программы.

## Заключение.

В результате выполнения данной лабораторной работы были реализованы:

* Класс стека – TStack.
* Реализована обратная польская запись.
* Реализованы операции: +, -, \*, /
* Расстановка приоритетов операций.
* Правильное использование скобок.

Разработан алгоритм и составлен программный комплекс для решения соответствующей задачи.



## Список используемой литературы.

* Гергель В.П. Рабочие материалы к учебному курсу «Методы программирования», ННГУ, 2002. – 100 c.
* <https://ru.wikipedia.org/wiki/Обратная_польская_запись>

## Приложение.

*TStack.h*

#pragma once

#include <iostream>

using namespace std;

#define MemSize 100

class TStack

{

int Top; //верхушка стека

int \*pMem;

int Size;

public:

TStack();

TStack(const TStack &s);

TStack(int n);

~TStack(void);

int IsEmpty(void)const;

int IsFull(void)const;

void Push(const int Val);

int Pop(void);

int Peek(void) const;

};

*Prot.h*

#include "TStack.h"

int Prioritet(char);

void Preobrazovanie(char \*const);

char Operation(char, char, char);

void Calc(char \*const);

bool Sravnenie(char, TStack);

*TStack.cpp*

#include "TStack.h"

TStack::TStack(const TStack &v)

{

Size = v.Size;

pMem = new int[Size];

Top = v.Top;

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

pMem[i] = v.pMem[i];

}

TStack::TStack()

{

Top = -1;

Size = MemSize;

pMem = new int[Size];

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

pMem[i] = 0;

}

TStack::TStack(int si) {

Top = -1;

Size = si;

pMem = new int[Size];

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

pMem[i] = 0;

}

TStack::~TStack()

{

delete[]pMem;

}

int TStack::IsEmpty() const

{

if (Top == -1)

{

return true;

}

else

{

return false;

}

}

int TStack::IsFull() const

{

if (Top == Size - 1)

{

return true;

}

else

{

return false;

}

}

void TStack::Push(int Val)

{

if (this->IsFull())

{

throw "IsFuul!";

}

Top++;

pMem[Top] = Val;

}

int TStack::Pop(void)

{

if (this->IsEmpty())

{

throw "IsEmpty";

}

Top--;

return pMem[Top + 1];

}

int TStack::Peek() const

{

return pMem[Top];

}

*Alg.cpp*

#include "TStack.h"

#include "Prot.h"

const int Max\_Size\_String = 100;

int Prioritet(char s) // приоритет операций

{

if (s == '(')

return 0;

if (s == ')')

return 1;

if ((s == '+') || (s == '-'))

return 2;

if ((s == '\*') || (s == '/'))

return 3;

else

{

return -1;

}

}

bool Sravnenie(char str, TStack A) //если приоритет пришедшешого оператора больше приоритета, оператора на вершине стека

{

char x;

x = A.Peek();

if (Prioritet(x) < Prioritet(str))

{

return true;

}

else

{

return false;

}

}

int Skobka(char \* str)

{

TStack s;

int array[Max\_Size\_String][2];

int n = 0;

int j = 0;

int m = 1;

int errors = 0;

int size = strlen(str);

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

{

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

{

s.Push(m++);

}

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

if (!s.IsEmpty())

{

n++;

array[j][0] = s.Pop();

array[j++][1] = m++;

}

else

{

n++;

array[j][0] = 0;

array[j++][1] = m++;

++errors;

}

}

while (!s.IsEmpty())

{

n++;

array[j][0] = s.Pop();

array[j++][1] = 0;

errors++;

}

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

{

cout << array[i][0];

cout << " " << array[i][1] << "\n";

}

return errors;

}

void Preobrazovanie(char str[])

{

int err = 0;

char y, x;

TStack A; //стек

int i = 0;

int z = 0; // индекс для res\_str

char res\_str[Max\_Size\_String] = "";

bool brackets = true;

err = Skobka(str);

cout << "OPZ:" << endl;

while (str[i] != '\0')

{

if (isdigit(str[i])) {

cout << str[i] << " ";

x = str[i];

res\_str[z] = x;

z++;

}

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

if (Sravnenie(str[i], A))

A.Push(str[i]);

else {

while (!Sravnenie(str[i], A)) {

x = A.Pop();

cout << x << " ";

res\_str[z] = x;

z++;

}

A.Push(str[i]);

}

}

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

A.Push(str[i]);

if ((str[i] == ')')) {

x = A.Peek();

while (x != '(') {

y = A.Pop();

cout << y << " ";

res\_str[z] = y;

z++;

x = A.Peek();

}

if (x != '(' || A.IsEmpty()) {

cout << "Error! " << endl;

brackets = false;

//break;

}

else {

A.Pop();

}

}

i++;

}

while (!A.IsEmpty()) {

x = A.Pop();

cout << x << " ";

res\_str[z] = x;

z++;

}

if (brackets)

Calc(res\_str);

else cout << "Error! " << endl;

}

char Operation(char str, char Op1, char Op2)

{

char result = 0;

if (str == '\*')

{

result = Op1\*Op2;

}

if (str == '/')

{

result = Op1 / Op2;

}

if (str == '+')

{

result = Op1 + Op2;

}

if (str == '-')

{

result = Op1 - Op2;

}

return result;

}

void Calc(char str[])

{

TStack A; //числа

int Operand1 = 0;

int Operand2 = 0;

char VAlue = 0;

char LastOperation = 0;

int Result = 0;

bool devine = false;

bool stack\_is\_empty = false;

int result = 0;

for (int i = 0; i < strlen(str); i++)

{

if (isdigit(str[i]))

A.Push(str[i]);

else {

VAlue = NULL;

VAlue = A.Pop();

if ((VAlue >= '0') && (VAlue <= '9'))

Operand2 = (int)VAlue - 48;

else

Operand2 = VAlue;

if (A.IsEmpty()) {

stack\_is\_empty = true;//не достаточно операндов

break;

}

else {

VAlue = NULL;

VAlue = A.Pop();

if ((VAlue >= '0') && (VAlue <= '9'))

Operand1 = (int)VAlue - 48;

else

Operand1 = VAlue;

LastOperation = str[i];

if (LastOperation == '/'&& Operand2 == 0)

{

devine = true;

break;

}

else {

result = Operation(LastOperation, Operand1, Operand2);

A.Push(result);

}

}

}

}

if (devine)

cout << "Error! " << endl;

else {

if (stack\_is\_empty) {

cout << "Error! " << endl;

}

else { cout << endl << "Resultat= " << result << endl; }

}

}

*Main.cpp*

#include "TStack.h"

#include "Prot.h"

const int Max\_Size\_String = 100;

int main()

{

char str[Max\_Size\_String] = "";

cout << "Vvedite virazhenie: ";

cin >> str;

Preobrazovanie(str);

cout << endl;

return 0;

}