Министерство образования и науки Российской федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт ИТММ

Отчет

по учебной практике

**Вычисление арифметических выражений.**

Выполнила:

студентка гр. 381703-1

Мочалова А.Р.

Проверил:

Волокитин В.Д.

г. Нижний Новгород 2018г

Оглавление

[Введение 3](#_Toc515272243)

[Постановка учебно-практической задачи 4](#_Toc515272244)

[Руководство пользователя 5](#_Toc515272245)

[Руководство программиста 7](#_Toc515272246)

[Заключение 10](#_Toc515272247)

[Список литературы 10](#_Toc515272248)

[Приложения 11](#_Toc515272249)

# Введение.

Разработка программы для вычисления арифметических выражений обычно включает в себя разработку обратной польской записи

****Обра́тная по́льская запись (**[**англ.**](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)***Reverse Polish notation, RPN*)** — форма записи [математических](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) и логических выражений, в которой [операнды](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D0%BD%D0%B4) расположены перед знаками [операций](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Также именуется как *обратная польская запись*, *обратная бесскобочная запись*, *постфиксная нотация*, *бесскобочная символика Лукасевича*, *польская инверсная запись*, *ПОЛИЗ*.

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.

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 (например, записав вместо выражения − 3 выражение 0 − 3), 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.

# Постановка учебно-практической задачи

**Цель работы:**

Разработать программу, выполняющую вычисление арифметического выражения с вещественными числами. Выражение в качестве операндов может содержать вещественные числа. Допустимые операции известны: +, -, /, \*. Допускается наличие знака "-" в начале выражения или после открывающей скобки. Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю вид ошибки и номера символов строки, в которых были найдены ошибки.

**Последовательность выполнения работы**

1. Разработка шаблонного класса TStack.
2. Разбиение исходного арифметического выражения на лексемы (т.е. выделить операнды, операции, переменные).
3. Проверка корректности выражения:

* правильность расстановки скобок (с использованием стека (TStack) для хранения индексов скобок).
* пропущены ли операнды или знаки операций
* недопустимые символы

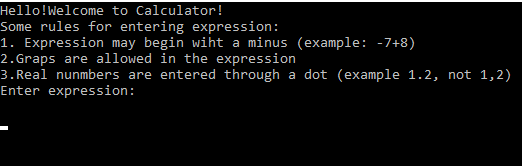
1. Перевод выражения в постфиксную (польскую) запись.
2. Вычисление выражения по постфиксной записи.
3. Создание консольного интерфейса пользователя.

# Руководство пользователя

После запуска программы на экране появится диалоговое окно. В нем описаны основные правила записи вычисляемого выражения, а именно:

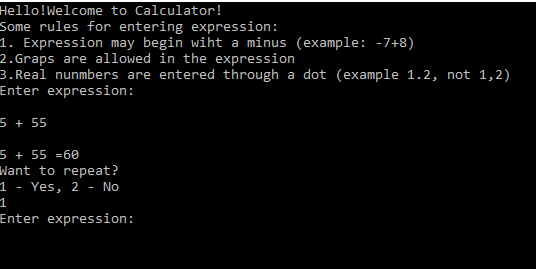
* Выражение может начинаться с минуса (программа учитывает унарный минус)
* Выражение может содержать пробелы
* Вещественные числа необходимо писать через точку, не через запятую

Затем пользователю предлагается ввести выражение.



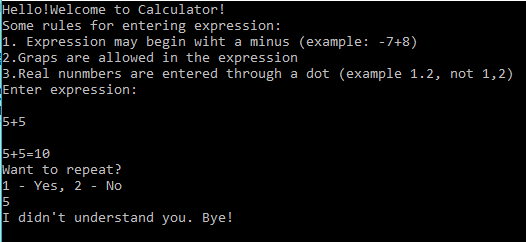
После ввода выражения на экране появится ответ.

Затем пользователю будет предложена возможность повторить вычисление.

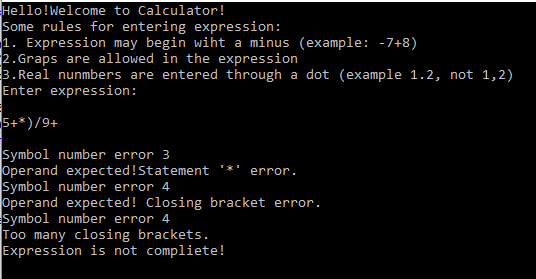


При выборе первого пункта у пользователя запрашивается новое выражение. При выборе второго пункта программа попрощается с пользователем и закроется.

Если пользователь введет что-то кроме 1 или 2, то программа выдаст сообщение об ошибке и закроется.



Также программа будет реагировать на ошибки в веденном выражении. На экране пользователю представится информация о символе, в котором допущена ошибка, а также о виде ошибки. Например, при вводе выражения «25+\*)/9+» программа известит пользователя о том, что отсутствует число между «+» и «\*» и «\*» «)», о том, что закрывающих скобок слишком много, а также известит пользователя о том, что выражение не завершено.



# Руководство программиста

Решение поставленной задачи осуществляется с помощью следующих элементов:

1.Шаблонный класс TStack

Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

В нем есть следующие поля:

* Массив значений
* Размер стека
* Количество элементов в стеке

Реализованы следующие методы:

* Проверка на пустоту
* Проверка стека на полноту
* Положить элемент (если стек заполнен, то перевыделяется память)
* Взять элемент

В качестве элемента стека могут выступать значения любых типов данных.

2. Структура Lex

В данной структуре хранятся лексемы. В ней есть следующие поля:

* OpType – тип лексемы, 1 , если это число, 2 , если это операция
* Value – значение лексемы, если это число
* Op – значение лексемы, если это операция
* Priority – приоритет лексемы, если это операция

3. Класс Computer

В данном классе в качестве полей записаны стеки Val и Oper типа Lex, использующиеся при создании обратной польской записи.

Реализованы следующие методы:

* Check

На вход в данный метод поступает строка, которая содержит выражение пользователя, и символ, который может считаться концом формулы («=»).

Затем программа идет по символам строки, считывает лексемы, проверяет основные правила записи арифметического выражения и, если нашлись ошибки, вызывает функцию Error. После того, как проверена вся строка, если в выражении присутствовали ошибки, данный метод вызывает исключение.

* Parse

На вход в данный метод поступает строка, которая содержит выражение пользователя, и символ, который может считаться концом формулы («=»).

Затем программа идет по символам строки и заполняет стек Val обратной польской записью по следующему алгоритму:

1. Если символ это цифра, то вызывается функция перевода строки в вещественное число, в стек Val заносится лексема числа с соответствующим значением.
2. Если это операция «+», то запоминается лексема операции со значением «+» и приоритетом 1.
3. Если это операция «\*» или «/», то запоминается лексема операции с соответствующим значением и приоритетом 2.
4. Если это операция «-« то осуществляется проверка на унарный минус, а именно: если перед минусом стоит не число, то запоминается лексема операции «-« с приоритетом 3. Если же минус не унарный, то запоминается лексема операции «-« с приоритетом 1.
5. Если это операция «(« , то в стек Oper заносится лексема операции «(« с приоритетом 0.
6. Если это операция «)», то из стека Oper все значения переносятся в стек Val , до тех пор, пока не дойдет до значения операции «(». После переноса, операция «(« удаляется из стека Oper.
7. Если выполнились пункты 2-4, то из стека Oper (если он не пустой) получается значение лексемы предыдущей операции. Затем сравнивают приоритеты текущей и прошлой операции. Если приоритет прошлой операции выше, то она заносится в стек Val , если нет, то в стек Oper . Затем лексема текущей операции помещается в стек Oper .
8. Переход на следующий символ строки.
9. После обработки всей строки, все операции из стека Oper переносятся в стек Val.
10. В конце работы метода в стеке Val находится обратная польская запись выражения пользователя.

* Compute

Данный метод использует стек Val, стек лексем In, в котором лежит «перевернутый» стек Val и собственный стек Res, в котором лежат промежуточные значения вычислений

Само вычисление происходит по следующему алгоритму:

До тех пор, пока стек In не пустой:

* 1. Из стека In получается лексема.
  2. Если это число, то оно заносится в стек Res.
  3. Если это операция «\*», то из стека Res берется два значения a и b , и выполняется a\*b.
  4. Если это операция «/», то из стека Res берется два значения a и b , и выполняется a/b. (Учитывается проверка того, что не 0)
  5. Если это операция «+», то из стека Res берется два значения a и b , и выполняется a+b.
  6. Если это операция «-« и он унарный (приоритет равен 3) то из стека Res берется одно значение и вычитается из 0. Если операция не унарная (приоритет равен 1), то из стека Res берется два значения a и b , и выполняется a - b.
  7. Результат выполнения пунктов 3-6 кладется в стек Res.

Когда стек In станет пустым, то в стеке Res будет лежать всего одно значение, это и есть результат вычисления всего выражения пользователя.

4. Функция Error

Данная функция принимает номер символа, в котором допущена ошибка, и номер ошибки.

Она выводит на экран соответствующее сообщение (все виды ошибок см. в приложении)

# Заключение

В данной лабораторной работе были изучены следующие аспекты:

1. Концепция стека и его реализация.
2. Описание лексем как структуры и разбор строки на лексемы.
3. Принцип проверки арифметического выражения на корректность и реализация соответствующей функции.
4. Разбор арифметического выражения в обратную польскую запись и реализация соответствующей функции.
5. Вычисление выражения на основе обратной польской записи и реализация соответствующей функции.
6. Принцип работы со строками.
7. Реализован диалог с пользователем

В ходе работы был разработан примитивный калькулятор, который учитывает корректность введенного выражения.

# Список литературы

# <https://www.codeproject.com/Articles/345888/How-to-write-a-simple-interpreter-in-JavaScript>

# <https://habr.com/post/100869/>

# <https://habr.com/post/282379/>

# Приложения

Файл stack.h:

Приложение 1. Разработка класса Stack и его методов

class TStack

{

private :

T \*mas; //массив элементов стека

int size; //размер стека

int quant; // количество элементов стека

public:

TStack(){size = 0; quant = 0; mas = new T[1];}

TStack(int s) //конструктор

{

if( s > 0)

{

quant = 0;

size = s;

mas = new T [s];

}

else throw "Negative length!";

}

int GetSize() {return size;} // возвращение размера стека

bool IsEmpty() // проверка на пустоту

{

if (quant == 0)

return 1;

return 0;

}

bool IsFull () // проверка на полноту

{

if ( quant == size )

{

return 1;

}

return 0;

}

void Push(T El) // положить элемент в стек

{

if (IsFull ())

{

T \*tmp = new T [size+10];

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

{

tmp [i] = mas[i];

}

delete []mas;

mas=tmp;

size = size + 10;

}

mas[quant] = El;

quant ++;

}

T pop() // взять эелемент из стека

{

if (IsEmpty() == 0)

{

T tmp = mas[quant-1];

quant --;

return tmp;

}

else throw ( " empty");

}

void Print() //вывод стека на экран

{

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

{

cout << mas[i] << " ";

}

cout << endl;

}

};

Файл arithmetic.h:

Приложение 2: Структура Lex

struct Lex

{

int OpType; // 1-число,2-операция

double Value;

char \*Op;

int Priority;

};

Приложение 3: функция Error для вывода сообщений об ошибке

void Error(int i, int j)

{

switch (j)

{

case 1:

cout << "Symbol number error " << i <<endl;

cout << "Opertion is expected! Opening bracket error." << endl;

break;

case 2:

cout << "Symbol number error " << i <<endl;

cout << "Operand expected! Closing bracket error." << endl;

break;

case 3:

cout << "Symbol number error " << i <<endl;

cout << "Too many closing brackets." << endl;

break;

case 4:

cout << "Symbol number error " << i <<endl;

cout << "Operand expected!Statement '+' error." << endl;

break;

case 5:

cout << "Symbol number error " << i <<endl;

cout << "Operand expected!Statement '-' error." << endl;

break;

case 6:

cout << "Symbol number error " << i <<endl;

cout << "Operand expected!Statement '\*' error." << endl;

break;

case 7:

cout << "Symbol number error " << i <<endl;

cout << "Operand expected!Statement '/' error." << endl;

break;

case 8:

cout << "Symbol number error " << i <<endl;

cout << "Unresolved symbol!" << endl;

break;

case 9:

cout << "Symbol number error " << i <<endl;

cout << "Operation is expected!" << endl;

break;

case 10:

cout << "Expression is not compliete!" << endl;

break;

case 11:

cout << "Too many opening brakets!" << endl;

break;

case 12:

cout << "Symbol number error " << i <<endl;

cout << "Expression inside brackets is expected!"<< endl;

default:

break;

}

}

Приложение 4: метод Check класса Computer, выполняющий предварительную проверку выражения.

void Check(char \*f, char end)

{

int Open = 0, i = 0, k = 0;

struct Lex lex1;

lex1.OpType = 0;

lex1.Op = " ";

while (\*f && (\*f != end))

{

switch (\*f)

{

case ' ':

case '\r':

case'\n':

f++;

i++;

continue;

case '(':

i++;

if ((lex1.OpType == 0)||(lex1.OpType == 2) && (lex1.Op != ")"))

{}

else {Error(i, 1); k++;/\*throw "Opertion is expected! Opening bracket error.";\*/}

Open++;

lex1.OpType = 2;

lex1.Op = "(";

f++;

break;

case ')':

i++;

if((lex1.OpType == 1) || ((lex1.Op == ")")))

{

}

else if (lex1.Op == "(")

{

Error(i, 12); k++;

}

else {Error(i, 2);k++;/\* throw "Operand expected! Closing bracket error.";\*/}

Open--;

lex1.OpType = 2;

lex1.Op = ")";

if (Open < 0)

{Error(i, 3); /\*throw "Too many closing brackets.";\*/}

f++;

break;

case '+':

i++;

if ((lex1.OpType == 1)||(lex1.Op == ")"))

{

}

else {Error(i, 4); k++;/\*throw "Operand expected!Statement '+' error.";\*/}

lex1.OpType = 2;

lex1.Op = "+";

f++;

break;

case '-':

i++;

lex1.OpType = 2;

lex1.Op = "-";

f++;

break;

case '\*':

i++;

if ((lex1.OpType == 1)||(lex1.Op == ")"))

{}

else {Error(i, 6);k++;/\*throw "Operand expected!Statement '\*' error.";\*/ }

lex1.OpType = 2;

lex1.Op = "\*";

f++;

break;

case '/':

i++;

if ((lex1.OpType == 1)||(lex1.Op == ")"))

{}

else {Error(i, 7);k++;/\*throw "Operand expected!Statement '/' error.";\*/}

lex1.OpType = 2;

lex1.Op = "/";

f++;

break;

default:

if (!strchr ("0123456789", \*f))

{i++; Error(i, 8);k++;/\*throw "Unresolved symbol!";\*/ f++; lex1.OpType = 4;}

else

{

if ((lex1.OpType == 0) ||(lex1.OpType == 2) && (lex1.Op != ")"))

{}

else {Error(i, 9);k++;/\*throw "Operation is expected!";\*/}

lex1.OpType = 1;

char\* a = f;

lex1.Value = strtod(f, &f);

i = i + (int)(f-a);

while ((strchr(" \n\r", \*f)) && (\*f) && (\*f != end))

{

f++;

i++;

}

}

break;

}

if ((\*(f) == end)||(!\*(f)) )

if ((lex1.Op != ")") && (lex1.OpType != 1))

{Error(i, 10);/\*throw "Expression is not compliete!";\*/ k++;}

}

if (Open >= 0 )

{Error(i, 11);/\*throw "Too many opening brakets!";\*/ k++;}

if (k !=0) throw "Errors in the expression recording!";

}

Приложение 5: метод Parse класса Computer разбирающее выражение в обратную польскую запись.

char \*Parse(char \*f, char end)

{

struct Lex lex;

lex.OpType = 0;

while (\*f && (\*f != end))

{

switch(\*f)

{

case ' ':

case '\r':

case'\n':

f++;

continue;

case '(':

lex.OpType = 2;

lex.Op = "(";

lex.Priority = 0;

f++;

Oper.Push(lex);

continue;

case ')':

f++;

if (!Oper.IsEmpty())

{

struct Lex lex2 = Oper.pop();

while(lex2.Op != "(")

{

Val.Push(lex2);

lex2 = Oper.pop();

}

}

else throw "" ;

continue;

case '-':

if (lex.OpType != 1)

{

while ((strchr(" \n\r", \*f)) && (\*f) && (\*f != end)) f++;

if(strchr ("0123456789", \*(f+1)))

{

f++;

lex.OpType = 1;

lex.Value = (-1)\*strtod(f, &f);

while ((strchr(" \n\r", \*f)) && (\*f) && (\*f != end)) f++;

}

else

{

lex.OpType = 2;

lex.Op = "-";

lex.Priority = 3;

f++;

}

}

else

{

lex.OpType = 2;

lex.Op = "-";

lex.Priority = 1;

f++;

}

break;

case '+':

lex.OpType = 2;

lex.Op = "+";

lex.Priority = 1;

f++;

break;

case '\*':

lex.OpType = 2;

lex.Op = "\*";

lex.Priority = 2;

f++;

break;

case '/':

lex.OpType = 2;

lex.Op = "/";

lex.Priority = 2;

f++;

break;

default:

lex.OpType = 1;

lex.Value = strtod(f, &f);

while ((strchr(" \n\r", \*f)) && (\*f) && (\*f != end)) f++;

}

if (lex.OpType == 2)

{

struct Lex lex2;

if (!Oper.IsEmpty())

{

lex2 = Oper.pop();

if ( lex2.Priority >= lex.Priority)

Val.Push(lex2);

else Oper.Push(lex2);

}

Oper.Push(lex);

}

else Val.Push(lex);

}

while (!Oper.IsEmpty())

Val.Push(Oper.pop());

return f;

}

Приложение 7: метод Compute класса Computer, осуществляющей вычисление выражения на основе его обратной польской записи.

double Compute (){

double tmp,a,b, res;

TStack <struct Lex> In;

TStack <double> Res;

struct Lex lex;

res = 0;

while ( !Val.IsEmpty())

In.Push(Val.pop());

while( !In.IsEmpty())

{

lex = In.pop();

if (lex.OpType == 1)

Res.Push(lex.Value);

else

{

a = Res.pop();

switch (lex.Op[0])

{

case '+':

b = Res.pop();

tmp =b + a;

break;

case '-':

if (lex.Priority == 1)

{

if (Res.IsEmpty())

b = 0;

else b = Res.pop();

tmp = b - a;

}

else

{

tmp = 0 - a;

}

break;

case '\*':

b = Res.pop();

tmp = b \* a;

break;

case '/':

b = Res.pop();

if (a != 0)

{

tmp = b/ a;

}

else throw "Division by zero is prohibited!";

break;

}

res= tmp;

Res.Push(res);

}

}

return Res.pop();

}

Файл main\_arithmetic.cpp:

Приложение 7: реализация интерфейса диалога с пользователем и запуска обработки выражений

int main()

{

double value;

int i;

char end = '=';

string af;

cout << "Hello!Welcome to Calculator!" << endl;

cout << "Some rules for entering expression:" << endl;

cout << "1. Expression may begin wiht a minus (example: -7+8)" << endl;

cout << "2.Graps are allowed in the expression" << endl;

cout << "3.Real nunmbers are entered through a dot (example 1.2, not 1,2)" << endl;

l: cout << "Enter expression:" << endl;

cout << endl;

cin.sync();

getline(cin,af);

char \*func = new char[af.length()];

strcpy( func, af.c\_str() );

cout<< endl;

Computer a ;

a.Check(func,end);

a.Parse(func,end);

value = a.Compute();

cout << func<< "=" << value<< endl;

cout << "Want to repeat?"<< endl;

cout << "1 - Yes, 2 - No"<<endl;

cin >> i;

if (i == 1)

goto l;

else if (i == 2)

cout << "It is sad. Bye!";

else cout << "I didn't understand you. Bye!";

return 0;

}