ННГУ им. Н.И. Лобачевского

Отчёт по лабораторной работе

**Вычисление арифметических выражений**

**Выполнил:**

студент Боев Р.А.

гр. 382003-1

ИТММ ПМИ

Нижний Новгород

2021 г

## Содержание

[Введение 3](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661930)

[Постановка задачи 3](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661931)

[Руководство пользователя 3](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661932)

[Руководство программиста 4](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661933)

[Описание структуры программы 4](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661934)

[Описание структур данных 4](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661935)

[Описание алгоритмов 5](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661936)

[Заключение 7](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661937)

[Приложения](file:///C:\Users\79960\Downloads\Polyakova_otchet.docx#_Toc532661939) 8

## Введение

Некоторые арифметические выражения мы можем посчитать самостоятельно и быстро. Но есть такие, для вычисления которых потребуется очень много времени. Для этого создаются программы, выполняющие вычисления арифметических выражений.

В лабораторной работе приводится постановка задачи вычисления арифметических выражений с использованием стека, описание алгоритмов вычислений, а также дается описание программы и правил ее использования, прилагается текст программы и результаты выполнения подсчетов.

## 

## Постановка задачи

Необходимо разработать программу, выполняющую вычисление арифметического выражения с вещественными числами. Выражение в качестве операндов может содержать переменные и вещественные числа. Допустимые операции известны: +, -, /, \*. Допускается наличие знака "-" в начале выражения или после открывающей скобки. Опционально - наличие математических функций (sin, соs, ln, exp, и т.д.) Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю вид ошибки и номера символов строки, в которых были найдены ошибки.

## 

## Руководство пользователя

Пользователь вводит выражение по следующим правилам:

* Не допускается использование других символов, кроме «1234567890.,+-\*/»
* Скобки должны быть расставлены по правилам математики
* Нельзя ставить знак операции после знака операции, т.е. «1+/2”», исключение составляет унарный минус

Программа выдает пользователю результат в консоль

Пользователь может продолжить вводить выражения и получать результат, пока не выйдет из программы.

### Руководство программиста

## Описание структуры программы:

**Библиотеки, использующиеся в программе:**

* Iostream (Для ввода и вывода информации)
* Vector (Для использования STL Vector как структуры хранения данных)
* String (Для удобной работы со строками)

**Алгоритм программы реализован следующим образом:**

1. Пользователь вводит выражение, которое записывается в строку.
2. Производится проверка строки на соответствие правилам ввода.
3. Происходит перевод строки в вектор лексем, в котором каждый элемент – это отдельная лексема.
4. Далее, программа переводит вектор лексем в обратную польскую запись.
5. После перевода, производится подсчет выражения в обратной польской записи.
6. Результат выводится в консоль.

## Описание структур данных:

В программе используется класс Tlexeme (см. приложение 1) и класс TStack (приложение 2).

## Описание алгоритмов:

***Конвертирование в польскую запись Tlexeme (прилож. 3)***

Данный алгоритм основан на использовании стека. На вход алгоритма поступает строка символов, на выходе должна быть получена строка с постфиксной формой. Каждой операции и скобкам приписывается приоритет.

В таблице представлены приоритеты операций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ( | +,- | \*, / | ^ | sin |
| 0 | 1 | 2 | 3 | 4 |

Предполагается, что входная строка содержит синтаксически правильное выражение. Входная строка просматривается посимвольно слева направо до достижения конца строки. Операндами будем считать любую последовательность символов входной строки, не совпадающую со знаками определённых в таблице операций. Операнды по мере их появления переписываются в выходную строку. При появлении во входной строке операции, происходит вычисление приоритета данной операции. Знак данной операции помещается в стек, если:

* Приоритет операции равен 0 (это « ( » ),
* Приоритет операции строго больше приоритета операции, лежащей на вершине стека,
* Стек пуст.

В противном случае из стека извлекаются все знаки операций с приоритетом больше или равным приоритету текущей операции. Они переписываются в выходную строку, после чего знак текущей операции помещается в стек. Имеется особенность в обработке закрывающей скобки. Появление закрывающей скобки во входной строке приводит к выталкиванию и записи в выходную строку всех знаков операций до появления открывающей скобки. Открывающая скобка из стека выталкивается, но в выходную строку не записывается. Таким образом, ни открывающая, ни закрывающая скобки в выходную строку не попадают. После просмотра всей входной строки происходит последовательное извлечение всех элементов стека с одновременной записью знаков операций, извлекаемых из стека, в выходную строку.

***Вычисление Tlexeme (приложение 4)***

Для подсчета нам понадобится стек.

1. В цикле проходим по массиву, в котором хранится обратная польская запись
2. Если элемент – число, то вставляем в стек это число
3. Если элемент – унарный минус, то берем из стека последнее значение, умножаем его на -1 и помещаем обратно в стек
4. Если все предыдущие условия не выполнились, то
   1. Берем из стека два операнда, не забыв, что сначала из стека возвращается второй операнд, а потом первый.
   2. Если текущая операция – плюс, то складываем операнды. Если бинарный минус – то вычитаем из первого операнда второй, если знак умножения, то умножаем операнды, если знак деления, то делим первый операнд на второй, не забыв проверить, что второй операнд не равен нулю.
   3. Помещаем в стек результат операции.
5. Ответом будет являться значение, которое останется в стеке.

***Проверка скобок в строке (Приложение 5)***

1. Если после открывающейся скобки идет знак бинарной операции или число, то выбрасываем исключение.
2. Если перед открывающейся скобкой идет число, то выбрасываем исключение.
3. Для определения правильности подсчета скобок, мы при нахождении в строке открывающейся скобки инкрементируем счетчик, а при нахождении закрывающейся скобки декрементируем его. Если когда-нибудь счетчик опустится ниже нуля, то выбрасываем исключение.

***Проверка строки на запрещенные символы***

Для проверки нам понадобится строка, в которой хранятся все позволенные символы.

Проходя по всей строке, мы сравниваем каждый символ строки со всеми символами строки позволенных символов. Если хотя бы один символ исходной строки не найдется в строке дозволенных символов, то строка будет признана неверной.

***Общая проверка остальных случаев***

Если в строке на первом месте находится бинарная операция, то выбрасываем исключение.

Также, последним символом в строке является операция, но не закрывающаяся скобка, то выбрасываем исключение.

### 

### Заключение

Лабораторная работа на тему «Вычисление арифметических выражений» дала возможность с практической точки зрения узнать, что такое стек. Также были получены базовые знания в такой теме как «разбор выражения». Получена практика проектирования классов и повторения общих знаний объектно-ориентированного программирования. Другие люди могут воспользоваться этой программой, чтобы подсчитать результат довольно простых выражений. Работа программы протестирована с помощью Google Tests

### Приложения

## Приложение 1 (Class Tlexeme)

class Tlexeme

{protected:

int Size;

string\* mLexeme; // массив строк разитых на лексемы

int\* priority; // приоритет лексем

int Flag = -1; // флаг обработки

public:

//Конструкторы и деструкторы

Tlexeme(string a="0");

~Tlexeme();

//Операции

void Fill(string a); //Загрузить новую строку

int Check\_correct(); //Проверка на корректность введения записи

string Pol(); // Вернуть польскую строку

double Calculation(int x=0); // Вычисление

//Перегрузка операций ввода/вывода

friend istream& operator>>(istream& in, Tlexeme& l)

{ in >> l.mLexeme[0];

string a = l.mLexeme[0];

l.Size = (l.mLexeme[0]).length();

l.mLexeme = new string[l.Size];

l.priority = new int[l.Size];

l. mLexeme[0] = a;

return in;

}

friend ostream& operator<<(ostream& out, const Tlexeme& l)

{ for (int i = 0; i < l.Size; i++)

{ out << l.mLexeme[i] << ' ';}

cout << endl;

/\*for (int i = 0; i < l.Size; i++)

{ out << l.priority[i] << ' ';}

cout << endl;\*/

return out;

}

private:

// Закрытые функции

void Divide(); //Разбить на лексемы

void Pol\_1(); //Конвертировать в польскую запись

int Check\_brack();//Проверка скобок

int Check\_sign(); //Проверка знаков операций

int Check\_point(); //Проверка точек в дробных числах

int Check\_variable();//Проверка переменных

int Check\_function();//Проверка функции

};

#endif

## Приложение 2 (Class TStack)

template <class StackType>

class TStack

{protected:

int Head;

int Size;

StackType\* mStack;

void Overcrowded(); // Проверка на переполнение

public:

//Конструкторы и деструкторы

TStack();

~TStack();

//Операции со стеком

void Push(StackType i); // Вставка элемента

StackType Pop(); // Извлечение элемента

StackType Top(); // Просмотр верхнего элемента (без удаления)

bool Empty(); // Проверка на пустоту

int Count(); // Получение количества элементов в стеке

void Clear(); // Очистка стека

};

## Приложение 3

void Tlexeme::Pol\_1()

{ if (!Check\_correct()) { throw 1; } else{

if(Flag!=2)

{ int Nbrack=0;

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

{ if (priority[i] == 4) { Nbrack++; };

if (priority[i] == 5) { priority[i] = 1; };

}

string\* mLexeme1; mLexeme1 = mLexeme; mLexeme = new string[Size- Nbrack];

int\* priority1; priority1 = priority; priority = new int[Size- Nbrack];

TStack <int> S1;

int j=0;

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

{

if ((priority1[i]==2)||(priority1[i] == 3))

{ mLexeme[j] = mLexeme1[i];

priority[j] = priority1[i];

j++;}

else

{ if (priority1[i]==1)

{ if (S1.Empty()) { S1.Push(i); }

else

{ int pr1 = PR(mLexeme1[i]), pr2= PR(mLexeme1[S1.Top()]);

if (pr1>pr2) { S1.Push(i); }

else

{ while ((pr1<=pr2)&&(!S1.Empty()))

{ int q=S1.Pop();

mLexeme[j] = mLexeme1[q];

priority[j] = priority1[q];

j++;

if(!S1.Empty()){ pr2= PR(mLexeme1[S1.Top()]);}

} S1.Push(i);

}

}

}

else

{ if (mLexeme1[i] == "(") { S1.Push(i); }

else

{ if (mLexeme1[i] == ")")

{ int w = 0;

while ((!S1.Empty()) && (w == 0))

{ int q = S1.Top();

if (mLexeme1[q] == "(") { w = 1; }

else

{ q = S1.Pop();

mLexeme[j] = mLexeme1[q];

priority[j] = priority1[q];

j++;

}

} S1.Pop();

}

}

}

}

}

while (!S1.Empty())

{ int q = S1.Pop(); mLexeme[j] = mLexeme1[q]; priority[j] = priority1[q]; j++;}

Size = Size - Nbrack;

Flag = 2;

}}

}

## Приложение 4 (Вычисление Tlexeme)

double Tlexeme::Calculation(int x)

{ if(Flag!=2){Pol\_1(); }

TStack <double> S1;

int s = Size, j=0;

double\* M1 = new double[Size];;

string\* M2 = new string[Size];

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

{ if (priority[i]==2){ S1.Push(Convert(mLexeme[i])); }

if (priority[i]==3)

{ int w=0,f=-1;

while((w<j)&&(f==-1))

{

if (M2[j] == mLexeme[i])

{

f = i; S1.Push(f);

}

w++;

}

if(f==-1)

{ double q;

cout<<"\tEnter the value of the variable "<< mLexeme[i]<<":\n\t";

if (x == 0) { cin >> q; }else{q=x;} S1.Push(q); M1[j] = q; M2[j] = mLexeme[i];

}

}

if (priority[i]==1)

{ double x,y;

x=S1.Pop();

if(mLexeme[i]=="sin"){S1.Push(sin(x));}

else

{ y=S1.Pop();

if(mLexeme[i]=="+"){ S1.Push(x+y); }

if(mLexeme[i]=="-"){ S1.Push(y-x); }

if(mLexeme[i]=="\*"){ S1.Push(x\*y); }

if(mLexeme[i]=="/"){ S1.Push(y/x); }

if(mLexeme[i]=="^"){ S1.Push(pow(y,x)); }

}

}

}

return S1.Pop();

}

## Приложение 5

int Tlexeme::Check\_brack()

{ //что-то стоит между двух скобок одинакового типа

int a = -1, i = 0;

while ((i < Size) && (a == -1))

{ if ((mLexeme[i]=="(")&&(mLexeme[i+1]!="(")&&(mLexeme[i+ 2]=="(")) { a = i; }

if ((mLexeme[i] == ")") && (mLexeme[i + 1] != ")") && (mLexeme[i + 2] == ")")) { a = i; }

i++;}

if (a == -1) //проверка на правильность растановки скобок

{ TStack <int> S1; i=0;

while ((i < Size) && (a == -1))

{ if(priority[i]==4)

{ if(mLexeme[i]=="("){ S1.Push(1); }

else

{ if(S1.Empty()){a=i;}

else {S1.Pop();}

}

} i++;

}

if(!(S1.Empty())){a=11;}

}

return a;

}