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.

Отчет

по учебной практике

**Вычисление арифметических выражений**

Выполнил:

студент гр. 381703-1

Поликсенов И.А.

Проверил:

Ассистент кафедры МОСТ ИИТММ

Волокитин В.Д.

г. Нижний Новгород

2018г

Оглавление

[Введение 3](#_Toc531969573)

[Постановка учебно-практической задачи 4](#_Toc531969574)

[Руководство пользователя 5](#_Toc531969575)

[Руководство программиста 6](#_Toc531969576)

[Описание структур данных 8](#_Toc531969577)

[Заключение 11](#_Toc531969578)

[Список литературы 12](#_Toc531969579)

[Приложения 13](#_Toc531969580)

[Приложение 1. Класс Stack 13](#_Toc531969581)

[Приложение 2. Класс TLexems 15](#_Toc531969582)

[Приложение 3. Файл arithmetic.cpp 15](#_Toc531969583)

# Введение

Вычисление арифметических выражений встречаются повсеместно на разном уровне.

Все отрасти современности так или иначе используют те или иные вычисления.

В данной работе, для вычисления арифметических операций будем использовать обратную польскую запись.

Обра́тная по́льская запись — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Также именуется как обратная польская запись, обратная бесскобочная запись, постфиксная нотация, бесскобочная символика Лукасевича, польская инверсная запись, ПОЛИЗ.

Обратная польская нотация была разработана австралийским философом и специалистом в области теории вычислительных машин Чарльзом Хэмблином в середине 1950-х на основе польской нотации, которая была предложена в 1920 году польским математиком Яном Лукасевичем.

# 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.
* 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

**Исходные данные**

|  |  |
| --- | --- |
| Str | Строка, содержащая арифметическое выражение, введенное пользователем. |

**Требуемый результат**

|  |  |
| --- | --- |
| Answer | Число, содержащее ответ на введенное выражение. |

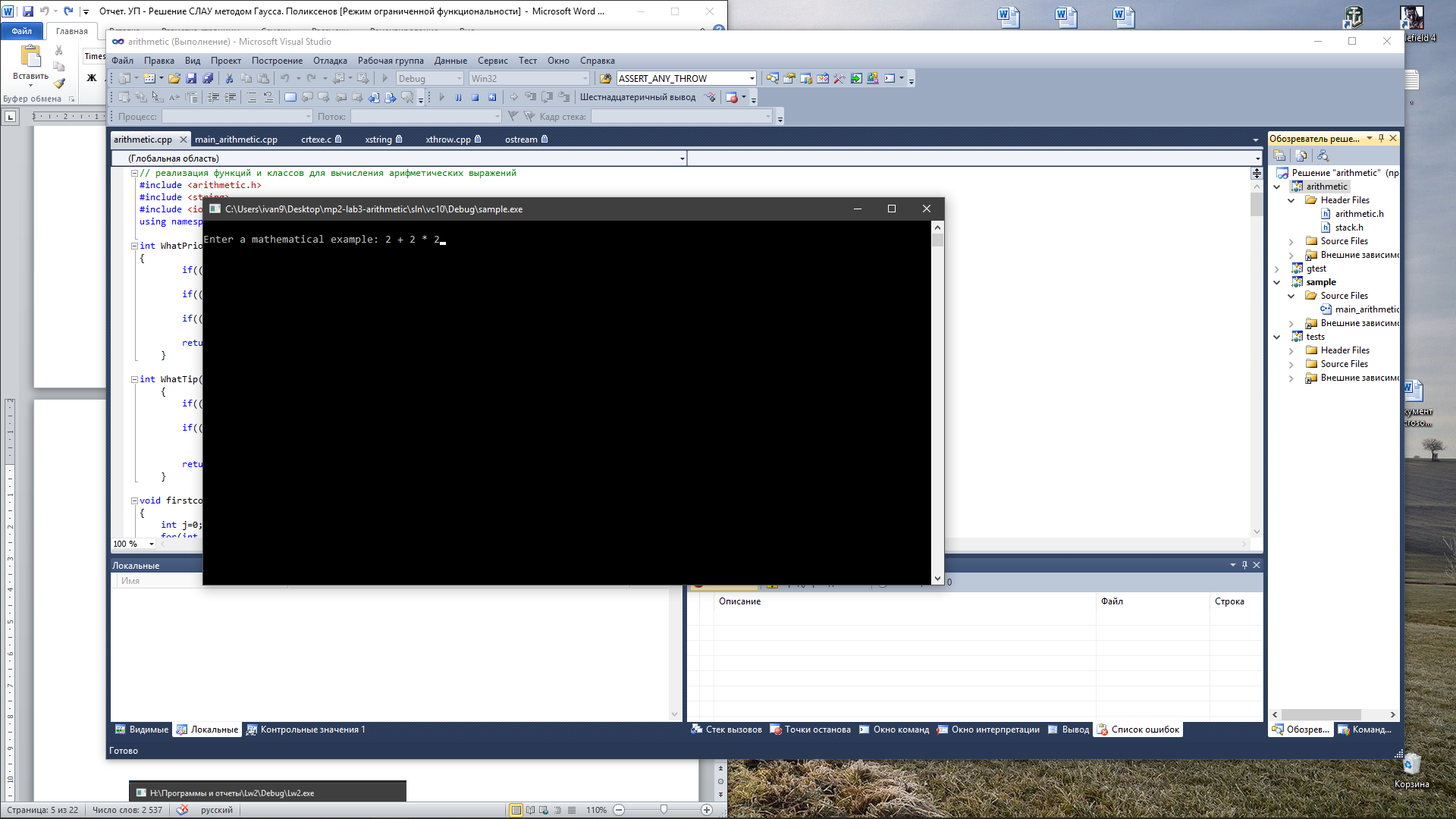
# Руководство пользователя

Для начала работы с программой необходимо открыть файл arithmetic.exe.

После запуска приложения на экране появится окно. Последует надпись, говорящая пользователю о необходимости ввода арифметического выражения. Пользователю необходимо ввести с клавиатуры арифметическое выражение, руководствуясь правилами ввода, и нажать Enter. (см. рис. 1)

Правила ввода:

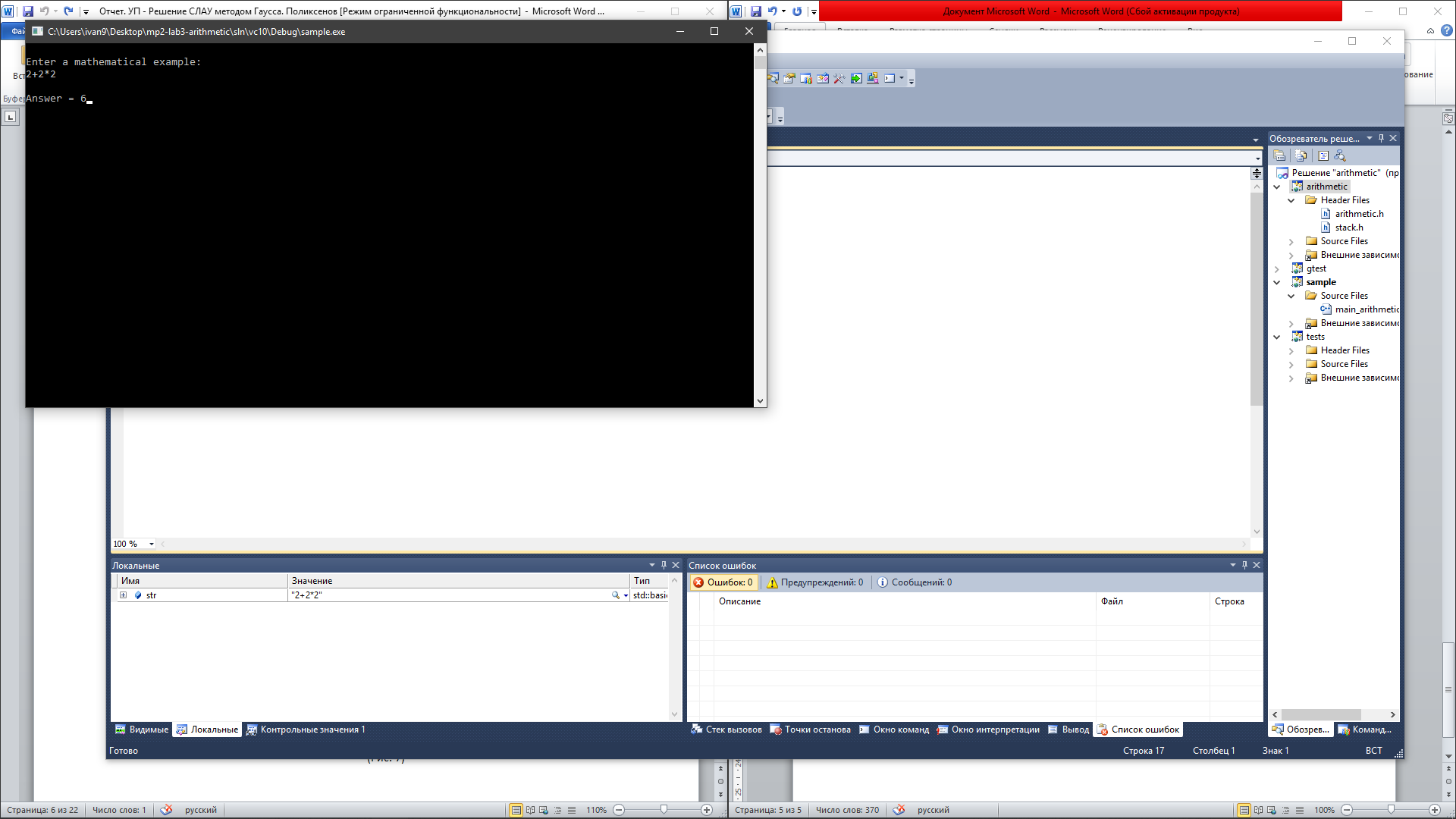
* 1. Правила оформления соответствуют стандартным;
  2. Допустимые символы: ‘+’ , ’ –‘ , ’ \*’ , ‘/’ , ’ (‘ , ’ )’ ;
  3. Реализован унарный минус;
  4. Вещественные числа вводить через точку.
  5. Допускаются пробелы между лексемами.



(рис.1)

Ввод арифметического выражения

После чего программа выведет ответ выражения на экран. (см. рис. 2)



(рис. 2)

Вывод ответа на экран

# Руководство программиста

Решение поставленной задачи осуществляется с помощью следующих функций.

**Функция первичной проверки**(void firstcontrol(string str) ).

Идя по строке, проверяет правильность ввода скобок по количеству. После этого проверяет каждый символ на правильность(принадлежность нашей системе).

**Функция вторичного контроля**(void secondcontrol(Stack<TLexems> res) ).

На вход поступает стек лексем.

Находит открывающую скобку, после чего ищет соответствующую ей закрывающую, если такая найдена – продолжает, иначе - выдает ошибку.

После чего проверяет каждую лексему на корректность ее ввода и расположения, исходя из основных правил арифметики.

**Функция перевода строки в стек лексем**(double interpreter(string str)).

В цикле движется от конца в начало, рассматривая каждый символ. Если соседние символы принадлежат одному типу лексем, объединяет их(для операций собственные настройки).

Учитывает возможность унарного минуса. Если унарный минус стоит перед числом, учитывает его путем записи отрицания этого числа.

**Выполнения алгоритма обратной польской записи**(double notation(Stack<TLexems> res)).

1. Берет очередной элемент, пока не рассмотрит все.

* Если это число – записывает его в стек результата.
* Если это открывающая скобка – записывает в дополнительный стек.
* Если это лексема – рассматривает приоритет последней записанной в дополнительный стек операции. Если больше или равно – выписывает из дополнительного стека все операции пока не встретит скобку или не станет пустым, и размещает рассматриваемую операцию. Если меньше – просто размещает.
* Если это закрывающая скобка – выписывает в стек результата все операции до открывающей скобки.

1. Если все элементы рассмотрены, но дополнительный стек не пустой – достаем все элементы в стек результата.

**Функция подсчета по обратной польской записи**(double calculate(Stack<TLexems> not))

* Инвертирует стек, полученный в ходе функции обратной польской записи.
* Проходит по каждому элементу стека.
* Если это число – помещает в дополнительный стек
* Если это операция – проверяет, какая она: унарная или бинарная. Исходя из полученной информации, достает из дополнительного стека два или одно число и выполняет необходимую операцию.
* Ответ помещает обратно в дополнительный стек.

По завершении вышеуказанного алгоритма, в дополнительном стеке остается одно число, которое и является нашим ответом.

**Функция определения приоритета**(int WhatPriority(string symbol)).

Приписывает каждой лексеме собственный приоритет.

«+» и «-» = 1;

«\*» и «/» = 3;

«(» и «)» = 5;

Для чисел, приоритет = 0.

**Функция определения типа**(int WhatTip(string symbol)).

Приписывает каждой лексеме собственный приоритет.

«+» и «-» и «\*» и «/» = 2; (Т.к. это бинарные операции. Если программа встречает унарный минус, она изменяет приоритет на месте).

«(» и «)» = 3;

Для чисел и точки, тип = 0.

# Описание структур данных

**Новые типы:**

**Class** **Stack** (реализует работу со стеком).

**Class TLexems** (Объединяет в один тип операции и операнды).

**Переменные:**

string str; Переменная, в которую пользователь записывает свое арифметической выражение.

Общее описание структуры програмного комплекса.

Решение поставленной задачи осуществляется с помощью многофайловой программы.

1. **Заголовочный файл stack.h**

В данном файле реализован следующий класс и методы:

template <class ValType>

class Stack

**Поля:**

ValType \*pStack; - хранилище элементов стека.

int finish; - индекс последнего занятого элемента.

int SizeE; - Количество элементов в стеке.

int SizeS; - Размер хранилища стека.

**Методы:**

1. ***Конструктор по умолчанию*** (Stack()).

Создает стек размера 2, с заполнением соответствующих параметров.

1. ***Конструктор с параметром***(Stack(int Size)).

Создает стек размера Size, с заполнением соответствующих параметров.

1. ***Очистка стека***(void ClrSt()).

Передвигает указатель на последний элемент в начало. Также сбрасывает значение количества элементов.

1. ***Удаление элемента стека***(void ClrElem()).

Передвигает указатель на соседний элемент. Также уменьшает значение количества элементов.

1. ***Получение количества элементов стека*** (int GetSizeE()).
2. ***Проверка на заполненность стека***(void IsFull()) .

Если полный – увеличить размер хранилища стека на указанный коэффициент DEFC.

1. ***Проверка на пустоту***(int IsEmpty())

Проверяет поле последнего занятого элемента. Если он меньше -1 – стек пустой, иначе – не пустой.

1. ***Взятие крайнего элемента стека*** (ValType Pop()).

Если стек не пустой – возвращает последний элемент, иначе выдает ошибку.

1. ***Размещение элемента на первое свободное место*** (void Push(ValType Elem)).

Размещает элемент типа ValType на первое свободное место. Если стек полный, увеличивает хранилище и помещает.

**2. Заголовочный файл arithmetic.h**

В данном файле реализован следующий класс и методы:

class TLexems

**Поля:**

string symbols; - содержит символ или набор символов лексемы;

int priority; - приоритет данного символа или набора символов;

int UnOrBinOrDig; - тип данного символа или набора символов.

**Методы:**

1. ***Конструктор с параметром***(TLexems (string sym))

Заполняет поле символа из входных данных. Для входного символа или набора символов определяет приоритет и тип.

***В файле прописаны следующие прототипы(Руководство к ним прописано в начале руководства программиста):***

int WhatPriority(string symbol); - прототип функции, определяющей приоритет строки.

int WhatTip(string symbol); - прототип функции, определяющий тип строки.

double interpreter(string Str); - прототип функции, разбивающей строку на лексемы.

void firstcontrol(string Str); - прототип функции, первичного контроля.

void secondcontrol(Stack<TLexems> res); - прототип функции, вторичного контроля.

double notation(Stack<TLexems> res); - прототип функции, осуществления записи в обратную польскую запись исходя из стека лексем.

double calculate(Stack<TLexems> not); - прототип функции, считающий выражение исходя из обратной польской записи.

**3. Файл arithmetic.cpp**

Реализованы все прототипы из файла arithmetic.h, а именно

1. int WhatPriority(string symbol);

string symbol – строка, введенная пользователем

1. int WhatTip(string symbol);

string symbol – строка, введенная пользователем

1. double interpreter(string Str);

string symbol – строка, введенная пользователем

1. void firstcontrol(string Str);

string symbol – строка, введенная пользователем

1. void secondcontrol(Stack<TLexems> res);

Stack<TLexems> res – стек, содержащий элементы типа TLexems. Результат работы функции interpreter.

1. double notation(Stack<TLexems> res);

Stack<TLexems> res – стек, содержащий элементы типа TLexems. Результат работы функции interpreter.

1. double calculate(Stack<TLexems> not)

Stack<TLexems> not - стек, содержащий элементы типа TLexems. Результат работы функции notation.

**4. Файл main\_arithmetic.cpp**

Основной файл программы. Реализует диалог с пользователем. Вызывает необходимые функции.

# Заключение

В данной лабораторной работе был изучен ряд известных алгоритмов, и был создан программный комплекс, реализующий следующие функциональные алгоритмы, касающиеся рассматриваемой темы:

1. Алгоритм обратной польской записи.
2. Алгоритм записи лексем из строки.
3. Алгоритм подсчета, исходя из обратной польской записи.
4. Ряд дополнительных функций, реализующих работу программы.
5. Класс стека и класс лексем.
6. Реализованы тесты для проверки вышеуказанных алгоритмов

В ходе работы удалось достичь правильного подсчета выражений с определенным набором операций.

# Список литературы

1. Алгоритм преобразования строки, введенной пользователем, в набор лексем [<https://www.codeproject.com/Articles/345888/How-to-write-a-simple-interpreter-in-JavaScript>]

# Приложения

## Приложение 1. Класс Stack

template <class ValType>

class Stack

{

public:

ValType \*pStack;

int finish;//Последний занятый

int SizeE;//Количество элементов

int SizeS;//Размер стека

Stack()//Конструктор по умолчанию

{

pStack = new ValType[2];

finish=-1;

SizeS=2;

SizeE=0;

}

Stack(int Size)//Конструктор с параметром

{

if(Size<0)

throw"Negative leinght";

pStack = new ValType[Size];

finish=-1;

SizeS=Size;

SizeE=0;

}

void ClrSt()//Очистка стека

{

finish=-1;

SizeE=0;

}

void ClrElem()//удаление крайнего элемента

{

finish--;

SizeE--;

}

ValType Check()//Проверка крайнего элемента

{

if(!(IsEmpty()))

return pStack[finish];

else

throw"stack is empty";

}

void IsFull()//Проверка на заполненность

{

if(finish==SizeS-1)

{

int NewSize = DEFC\*SizeS;

ValType \*pStack2;

pStack2 = new ValType[NewSize];

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

pStack2[i]=pStack[i];

delete[]pStack;

pStack=pStack2;

SizeS=NewSize;

}

}

int IsEmpty()//Проверка на пустоту

{

if(finish==-1)

return 1;

else

return 0;

}

ValType Pop()//Достает крайний элемент

{

if(!(IsEmpty()))

{

ValType res = pStack[finish];

finish--;

SizeE--;

return res;

}

else

throw"stack is empty";

}

void Push(ValType Elem) //Помещает элемент в первое свободное место

{

IsFull();

finish++;

SizeE++;

pStack[finish]=Elem;

}

};

## Приложение 2. Класс TLexems

class TLexems

{

public:

string symbols;

int priority;

int UnOrBinOrDig;

TLexems (string sym) //Конструктор с параметром

{

symbols=sym;

priority = WhatPriority(sym);

UnOrBinOrDig = WhatTip(sym);

}

};

## Приложение 3. Файл arithmetic.cpp

int WhatPriority(string symbol)//Определяет приоритет строки

{

if((symbol=="+")||(symbol=="-"))

return 1;

if((symbol=="\*")||(symbol=="/"))

return 3;

if((symbol==")")||(symbol=="("))

return 5;

return 0;

}

int WhatTip(string symbol) //Определяет тип строки

{

if((symbol=="+")||(symbol=="-")||(symbol=="\*")||(symbol=="/"))

return 2;

if((symbol=="(")||(symbol==")"))

return 3;

return 0;

}

void firstcontrol(string str) //первичный контроль

{

int j=0;

for(int i=0; i<str.length(); i++)//проверка соответствия скобок

{

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

j++;

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

j--;

if((i!=str.length()-1)&&(str[i]=='(')&&(str[i+1]==')'))

{

cout<<"errors symbol "<<i;

throw"empty between parentheses";

}

if((i!=str.length()-1)&&(str[i]==')')&&(str[i+1]=='('))

{

throw"empty between parentheses";

cout<<"errors symbol "<<i;

}

}

if(j>0)

throw"missing closing parentheses";

if(j<0)

throw"missing opening parentheses";

string a;

for(int i=0; i<str.length(); i++)//проверка некорректных символов

{

a=str.substr(i, 1);

switch (WhatTip(a))

{

case 2: break;

case 3: break;

case 0: if((str[i]<='9')&&(str[i]>='0')||(str[i]=='.')||(str[i]==' '))

break;

cout<<"errors symbol "<<i;

throw"incorrect symbols";

}

}

}

void secondcontrol(Stack<TLexems> res) //Вторичный контроль

{

int k,z=0, j, m=res.GetSizeE()-1;

for(int i=res.GetSizeE()-1; i>=0 ;i--)//проверка на соответствие скобок

{

k=0;

if(res.pStack[i].symbols=="(")

{

z=1;

for(j=m;j>=0;j--)

if((res.pStack[j].symbols==")")&&(j<i))

{

k=1;

m=j-1;

break;

}

}

if((!k)&&(z))

{

cout<<"errors lexems "<<i;

throw"incorrect parentheses";

}

z=0;

}

for(int i=0; i<res.GetSizeE();i++)

{

k=0;

switch (res.pStack[i].UnOrBinOrDig)

{

case 2: {

if((i==res.GetSizeE()-1)||(res.pStack[i-1].UnOrBinOrDig==2)||(res.pStack[i-1].symbols==")"))

{

cout<<"errors lexems "<<i-1;

throw"incorrect lexems";

}

}

break;

case 3: {

if(res.pStack[i].symbols=="(")

if(res.pStack[i-1].UnOrBinOrDig==2)

{

cout<<"errors lexems "<<i-1;

throw"incorrect symbols after opening parentheses";

}

if(res.pStack[i].symbols==")")

if((i!=0)&&(res.pStack[i-1].UnOrBinOrDig==0))

{

cout<<"errors lexems "<<i-1;

throw"incorrect symbols after closing parentheses";

}

}

break;

case 0: {

for(int l=0;l<res.pStack[i].symbols.length()-1;l++)//проверяем числа на подобную ошибку 2.2.1

if(res.pStack[i].symbols[l]=='.')

k++;

if(k>1)

{

cout<<"errors lexems "<<i;

throw"incorrect digit";

}

if((i!=0)&&((res.pStack[i-1].UnOrBinOrDig==0)||(res.pStack[i-1].symbols=="(")))

{

cout<<"errors lexems "<<i-1;

throw"incorrect symbols after digit";

}

}

break;

}

}

}

double interpreter(string str) //Интерпретатор

{

Stack<TLexems> res;

double answer=0.0;

int i = str.length()-1;

int k = 0, un = 0;

firstcontrol(str);

while(i >= 0)

{

un=0;

k = 0;

if((str.substr(i, 1)=="-")&&(str.substr(i+1, 1)=="("))//учитываем унарный минус

if((i==0)||((i!=0)&&(WhatTip(str.substr(i-1, 1))==2)))

{

k = 1;

i--;

un=1;

}

if(i>=0)

if ((WhatPriority(str.substr(i, 1))==0))//выделение чисел

{

k = 1;

i--;

while((i>=0)&&((WhatPriority(str.substr(i, 1))==0)||((str.substr(i, 1)=="-")&&((i==0)||(str.substr(i-1, 1)=="(")||(WhatTip(str.substr(i-1, 1))==2)))))//учитываем, что минус может стоять перед числом и это будет отриц. число

{

k++;

i--;

}

}

if ((i>=0)&&(WhatPriority(str.substr(i, 1))!=0)&&(k==0)&&(WhatPriority(str.substr(i, 1))!=5))//Выделение лексем

{

k = 1;

i--;

if(i>=0)

while((WhatPriority(str.substr(i, 1))!=0)&&(WhatPriority(str.substr(i, 1))!=5))

{

k++;

i--;

}

}

if ((i>=0)&&(WhatPriority(str.substr(i, 1))==5)&&(k==0))//выделение скобок

{

k = 1;

i--;

}

TLexems a(str.substr(i+1, k));

if(un==1)//учитываем унарный минус

{

a.UnOrBinOrDig=1;

a.priority=4;

}

//cout<<a.symbols<<" ";

res.Push(a);

}

secondcontrol(res);

return notation(res);

}

double notation(Stack<TLexems> res) //Создание нотации

{

int Size = res.GetSizeE();

Stack<TLexems> not(Size);

Stack<TLexems> lex;

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

{

switch(res.Check().priority)

{

case 0:not.Push(res.Pop());break;

case 5:

{

if(res.Check().symbols=="(")

lex.Push(res.Pop());

if(res.Check().symbols==")")

{

res.ClrElem();

while(lex.Check().priority!=5)

not.Push(lex.Pop());

lex.ClrElem();

}

}

break;

case 1:

{

if(lex.IsEmpty())

lex.Push(res.Pop());

else

{

if((lex.Check().priority<1)||(lex.Check().symbols=="("))

lex.Push(res.Pop());

else

{

while(!lex.IsEmpty())

{

if(lex.Check().symbols=="(")

break;

not.Push(lex.Pop());

}

lex.Push(res.Pop());

}

}

}

break;

case 4:

{

if(lex.IsEmpty())

lex.Push(res.Pop());

else

if((lex.Check().priority<4)||(lex.Check().symbols=="("))

lex.Push(res.Pop());

else

{

while(!lex.IsEmpty())

{

if(lex.Check().symbols=="(")

break;

not.Push(lex.Pop());

}

lex.Push(res.Pop());

}

}

break;

case 3:

{

if(lex.IsEmpty())

lex.Push(res.Pop());

else

if((lex.Check().priority<3)||(lex.Check().symbols=="("))

lex.Push(res.Pop());

else

{

while(!lex.IsEmpty())

{

if(lex.Check().symbols=="(")

break;

not.Push(lex.Pop());

}

lex.Push(res.Pop());

}

}

}

}

while(!lex.IsEmpty())

not.Push(lex.Pop());

cout<<endl;

return calculate(not);

}

double calculate(Stack<TLexems> not) //Вычисление выражения

{

int Size = not.GetSizeE();

Stack<TLexems> invert;

Stack<double> result;

while (!not.IsEmpty())

invert.Push(not.Pop());

string z;

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

{

if(invert.IsEmpty()!=1)

switch(invert.Check().priority)

{

case 0:

{

z=invert.Pop().symbols;

result.Push(atof(z.c\_str()));

}

break;

case 1:

{

if((invert.IsEmpty()!=1)&&(invert.Check().symbols=="+"))

{

invert.ClrElem();

double a= result.Pop();

double b= result.Pop();

result.Push(a+b);

}

if((invert.IsEmpty()!=1)&&(invert.Check().symbols=="-")&&(invert.Check().priority==1))

{

invert.ClrElem();

double a= result.Pop();

double b= result.Pop();

result.Push(b-a);

}

}

break;

case 4:

{

if((invert.IsEmpty()!=1)&&(invert.Check().symbols=="-"))

{

invert.ClrElem();

double a= result.Pop();

result.Push(-1\*a);

}

}

break;

case 3:

{

if((invert.IsEmpty()!=1)&&(invert.Check().symbols=="\*"))

{

invert.ClrElem();

double a= result.Pop();

double b= result.Pop();

result.Push(a\*b);

}

if((invert.IsEmpty()!=1)&&(invert.Check().symbols=="/"))

{

invert.ClrElem();

double a= result.Pop();

double b= result.Pop();

if(a==0)

throw"division by 0";

result.Push(b/a);

}

}

break;

}

}

double a = result.Pop();

return a;

}