Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, механики и математики

**Отчет по лабораторной работе**

**АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ**

**Выполнил**:студент группы 0826-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панов А. А.

Подпись

**Проверил**:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев С. А.

Подпись

Нижний Новгород

2016

**Содержание**

[Введение 3](#_Toc471377873)

[Польская нотация 3](#_Toc471377874)

[Теория автоматов 4](#_Toc471377875)

[Постановка задачи 5](#_Toc471377876)

[Руководство пользователя 6](#_Toc471377877)

[Руководство программиста 7](#_Toc471377878)

[Описание структуры программы 7](#_Toc471377879)

[Описание структур данных 8](#_Toc471377880)

[Описание алгоритмов 16](#_Toc471377881)

[Заключение 23](#_Toc471377882)

[Литература 24](#_Toc471377883)

[Приложение. Фрагменты исходного кода программы 25](#_Toc471377884)

[template <class Data> class Stack 25](#_Toc471377885)

[template <class Data> class List 27](#_Toc471377886)

[class Lexeme 28](#_Toc471377887)

[class LexemeNum: public Lexeme 29](#_Toc471377888)

[class LexemeVar: public Lexeme 30](#_Toc471377889)

[class LexemeOper: public Lexeme 31](#_Toc471377890)

[class UnaryOper: public LexemeOper 32](#_Toc471377891)

[Выделение лексем 33](#_Toc471377892)

[Перевод в польскую запись 37](#_Toc471377893)

[Вычисление 38](#_Toc471377894)

# Введение

Многие задачи требуют громоздких математических вычислений и приближенный ответ, для его получения можно использовать ЭВС. При этом хотелось бы использовать «привычный» вид записи арифметических выражений. Например, задача вычислить при различных x, y:

Но возникает проблема с порядком действий, расшифровкой записи и проверки её на корректность. Тогда для вычисления используется польская нотация, в которую переводится выражение, а проверка ввода может осуществляться с помощью теории автоматов.

## Польская нотация

*Польская нотация* (запись), также известна как префиксная нотация (запись), это форма записи [логических](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), [арифметических](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D0%BA%D0%B0) и [алгебраических](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) выражений. Характерная черта такой записи — [оператор](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%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)) располагается слева от [операндов](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D0%BD%D0%B4).

Перед переводом строки в польскую запись происходит разбиение на лексемы. Лексема — операнд или оператор. Любое арифметическое выражение можно разбить на лексемы. Например, выражение «sin(3\*x)+5» разбивается на лексемы:

* “sin”;
* “(” ;
* “3”;
* “\*”;
* “x” ;
* “)”;
* “+”;
* “5” ;

Выделим типы лексем:

* Число;
* Переменная;
* Бинарный оператор;
* Унарный оператор;
* Левая скобка
* Правая скобка

## Теория автоматов

*Теория автоматов*  — раздел [дискретной математики](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), изучающий [абстрактные автоматы](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82) — вычислительные машины, представленные в виде математических моделей — и задачи, которые они могут решать.

Разбиение строки на лексемы осуществляется некоторой системой. Происходит это по следующему принципу:

* Система находится в начальном состоянии state0.
* Считывается один символ.
* Ожидается, что он принадлежит типу 1 или 2 … или n.
* Создается лексема типа 1 или 2 … или n.
* Состояние переводится в состояние state1 или state2 … или state3.

Если следующий символ не принадлежит типам 1, 2, …, n то система сообщает об ошибке ввода.

# Постановка задачи

* Написать библиотеку позволяющую:
* Считать и вычислить арифметическое выражение с помощью польской нотации
* Выполнить проверку ввода
* Реализовать пример – калькулятор

Считываемое выражение может содержать:

* Вещественные операнды, переменные;
* Операторы:

+, -, \*, /, ^, cos, sin, tg, ( );

# Руководство пользователя

Пользователю предоставляется возможность:

* разбить строку s на лексемы с помощью функции NewRead(sting s, List <Lexeme\*> l) и записать их в список указателей на лексему l, при этом функция вернет число лексем numLex (если считывание будет неудачным – вызовется ошибка).
* получить польскую нотацию из l и записать её в стек из указателей на лексему с помощью функции GetPolishNotation(l, numLex) функция вернет указатель на стек Stack<Lexeme\*>
* Получить результат вычисления польской нотации с помощью функции double GetAnswer(Stack<Lexeme\*> \*pSt)
* Присвоить значения использованным переменным с помощью функции ReadVar()



* Очистить значения заполненных переменных с помощью статической функции Clear() класса LexemeVar

Рассмотрим [пример](#Сумма_ряда) создания программы считающей сумму n членов функционального ряда, по n-ому члену:

Пусть пользователь ввел n-ый член ряда

1. Считаем с помощью функции NewRead
2. Получим представление в польской записи с помощью функции GetPolishNotation
3. Заблокируем считывание переменной n с помощью функции Lexeme::blockVar(‘n’)
4. Считаем переменные с помощью функции ReadVar()
5. Для всех n от 1 до N найдем значение n-ого члена

# Руководство программиста

## Описание структуры программы

Программа состоит из пяти проектов:

* mp2-lab3-polish\_notation2 – статическая библиотека. В ней реализовано разбиение строки на лексемы, перевод в польскую запись и вычисление польской записи.

В проекте есть следующие классы:

* + [Lexeme](#Lexeme)
  + [LexemeVar](#LexemeVar)
  + [LexemeNum](#LexemeNum)
  + [LexemeOper](#LexemeOper)
  + [BinaryOper](#LexemeBinaryOper)
  + [UnaryOper](#LexemeUnaryOper)
  + [Template class Stack](#Stack)
  + [Template class List](#List)
* gtest – библиотека google тестов;
* test – тесты для классов и функций проекта mp2-lab3-polish\_notation2;
* example – пример простого калькулятора
* functionalSeries – пример вычисления суммы ряда

## Описание структур данных

#### Класс Stack<Data>

Шаблонный стек на массиве;

Поля класса:

protected:

Data \*ar;

int size;

int top;

##### Конструкторы и методы класса:

* Конструктор инициализации
* Деструктор
* Методы:
  + Data quick\_pop();
  + Data pop();
  + void quick\_push(Data num);
  + void push(Data num);
  + Data& ShowTop();
  + Data GetElem(int index);
  + bool isEmpty();
  + bool isFull();
* Перегруженные операторы:
  + <<

#### Структура Node<Data>

Поля структуры:

Data data;

Node\* next;

#### Класс List<Data>

Односвязный шаблонный список;

Поля класса:

protected:

Node <Data>\* head;

##### Конструкторы и методы класса:

* Конструктор по умолчанию
* Деструктор
* Методы:
  + bool isEmpty();
  + Data quick\_pop\_front();
  + void quick\_push\_front(Data num);
  + int reverce();
* Перегруженные операторы:
  + <<

#### Базовый класс Lexeme

Поля класса:

protected:

string name;

int type;

##### Конструкторы и методы класса:

* Конструктор инициализации
* Методы:
  + int getType()
  + virtual int getPriority() { return -1; }
  + virtual double getVal() { return -1; }
  + virtual double binaryCalculate(Lexeme \*l1, Lexeme \*l2) { return -1; }
  + virtual double unaryCalculate(Lexeme \*l1) { return -1; }
  + string getName() { return name; }
* Перегруженные операторы:
  + virtual bool operator >= (Lexeme& l)
  + Lexeme& operator=(const Lexeme& l)
  + bool operator==(const Lexeme &l)
  + bool operator!=(char s)
  + <<

#### Класс LexemeNum: public Lexeme

Поля класса:

private:

double val;

##### Конструкторы и методы класса:

* Конструктор инициализации
* Методы:
  + double getVal() { return val; }
* Перегруженные операторы:
  + bool operator==(int num);

#### Класс LexemeVar: public Lexeme

Поля класса:

public:

static double ar[26];

static bool isVar[26];

static int count;

##### Конструкторы и методы класса:

* Конструктор инициализации
* Методы:
  + double getVal()
  + static void clear()

#### Класс LexemeOper: public Lexeme

Поля класса:

protected:

int priority = lowest;

##### Конструкторы и методы класса:

* Конструктор инициализации
* Методы:
  + int getPriority()
* Перегруженные операторы:
  + Lexeme& operator=(const hLexeme& l)
  + bool operator >=(const Lexeme &l)

#### Класс UnaryOper: public LexemeOper

##### Конструкторы и методы класса:

* Конструктор инициализации
* Методы:
  + double unaryCalculate(Lexeme \*l1)

#### Класс BinaryOper: public LexemeOper

##### Конструкторы и методы класса:

* Конструктор инициализации
* Методы:
  + double binaryCalculate(Lexeme \*l1, Lexeme \*l2)

## Описание алгоритмов

#### Разбиение на лексемы

int NewRead(string &s, List <Lexeme\*>&l, int numLex = 0);

Функция возвращает число лексем в строке s, в случае неправильного ввода вызывает ошибку.

Таблица 1 Конечный автомат

|  |  |  |
| --- | --- | --- |
| Статус системы | Ожидаемый тип | Следующий статус |
| wait | num (число – «123») | next\_operator |
| var (переменная – «a, b, … z») | next\_operator |
| leftB (левая скобка) | wait |
| un\_op (унарный оператор «-») | next\_expression |
| funct (функция – «sin, cos, tg») | next\_expression |
| next\_operator | bin\_op («+ - \* /») | next\_expression |
| bin\_postf ( ^ ) | next\_op\_bin |
| next\_op\_bin | bin\_op («+ - \* /») | next\_expression |
| rightB (правая скобка) | next\_operator |
| next\_expression | num (число – «123») | next\_operator |
| var (переменная – «a, b, … z») | next\_operator |
| leftB (левая скобка) | wait |
| funct (функция – «sin, cos, tg») | wait |

#### Перевод в польскую запись

Таблица 2 Приоритеты

|  |  |  |
| --- | --- | --- |
| Тип оператора | Операторы | Приоритет |
| бинарный | «+», «-» | low |
| «\*», «/», «^» | medium |
| унарный | «-» | high |
| «sin», «cos», «tg» | high |

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. Реализован алгоритм, в котором функции являются унарными операторами с высоким приоритетом. В итоге получаем тот же результат.
* 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 символ является оператором о1, тогда:

1) пока приоритет o1 меньше приоритета оператора, находящегося на вершине стека…

… выталкиваем верхний элемент стека в выходную строку;

2) 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.

#### Вычисление

1. Обработка входного символа
   * Если на вход подан операнд, он помещается на вершину стека tmp.
   * Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека st, взятых в порядке добавления. Результат выполненной операции кладётся на вершину стека tmp.
2. Если входной набор символов обработан не полностью, перейти к шагу 1.
3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека tmp.

##### Класс Lexeme

Наиболее интересные методы класса

* bool operator==(const Lexeme &l);

return (type == l.type && name == l.name);

Возвращает true, если лексемы равны.

* virtual double getVal() { return -1; }

##### Класс LexemeNum

Наиболее интересные методы класса

* double getVal() { return num; }

##### Класс LexemeVar

Наиболее интересные методы класса

* double getVal()

{return ar[name[0]-'a'];}

В name[0] хранится имя переменной: a,b,…z

Массив ar[] – статическое поле класса LexemeVar в котором хранятся значения переменных.

Функция вернет значение переменной, которое задал пользователь.

* static void clear()

{

for (int i = 0; i < 26; i++)

{

ar[i] = 1;

isVar[i] = false;

}

}

Массив isVar[] – статическое поле класса LexemeVar, isVar[i] = true тогда и только тогда, когда i-ая переменная использована пользователем.

Функция удалит все переменные заданные пользователем.

##### Классы операторов

* [Унарный оператор](#UnaryOper)
* [Бинарный оператор](#BinaryOper)

# Заключение

В данной лабораторной работе вычислялись значения арифметических выражений. Для этого выражения разбивались на *лексемы* и проверялись на *корректность* ввода с помощью простейшего конечного автомата. Затем выполнялась необходимая сортировка лексем и итоговое вычисление.

# Литература

* Гергель В. П., Лабутина А. А. Учебно-образовательный комплекс по методам программирования //Нижний Новгород: ННГУ им. НИ Лобачевского. – 2007.

# Приложение. Фрагменты исходного кода программы

## template <class Data> class Stack

Stack(int \_size)//конструктор инициализации

{

if (\_size >= 0)

top = -1;

else throw NEGATIVE\_INDEX;

size = \_size;

ar = new Data[size];

}/\*-------------------------------------------------------------------------\*/

void push(Data num)

{

if (top + 1 < size)

ar[++top] = num;

else throw STACK\_FULL;

}/\*-------------------------------------------------------------------------\*/

void quickPush(Data num)

{

ar[++top] = num;

}/\*-------------------------------------------------------------------------\*/

Data pop()

{

if (top >= 0)

return ar[top--];

else throw STACK\_EMPTY;

}/\*-------------------------------------------------------------------------\*/

Data quickPop()

{

return ar[top--];

}/\*-------------------------------------------------------------------------\*/

Data& showTop()

{

return ar[top];

}/\*-------------------------------------------------------------------------\*/

Data getElem(int i)

{

return ar[i];

}/\*-------------------------------------------------------------------------\*/

bool isEmty()

{

return (top == -1);

}/\*-------------------------------------------------------------------------\*/

bool isFull()

{

return (top + 1 == size);

}/\*-------------------------------------------------------------------------\*/

int getNumTop()

{

return top;

}/\*-------------------------------------------------------------------------\*/

void setTop(int \_top)

{

top = \_top;

}/\*-------------------------------------------------------------------------\*/

~Stack()//деструктор

{

if (ar != 0)

delete[] ar;

}/\*-------------------------------------------------------------------------\*/

## template <class Data> class List

protected:

Node <Data> \*head;

public:

bool isEmpty() { return (head == NULL); }

Data quick\_pop\_front()

{

Node<Data> tmp = \*head;

Node<Data> \*p = head;

if (head->next)

{

head = head->next;

delete p;

}

else

{

delete head;

head = 0;

}

return tmp.data;

}/\*-------------------------------------------------------------------------\*/

void push\_front(Data obj)

{

Node<Data> \*newNode = new Node<Data>;

newNode->data = obj;

newNode->next = head;

head = newNode;

}/\*-------------------------------------------------------------------------\*/

int reverce()

{

int count = 0;

Node<Data> \*tmp = 0;

Node<Data> \*next = 0;

if (head)

next = head->next;

while (next != 0)

{

head->next = tmp;

tmp = head;

head = next;

next = head->next;

count++;

}

count = count + 1;

head->next = tmp;

return count;

}/\*-------------------------------------------------------------------------\*/

~List()

{

Node <Data> \*tmp;

while (!isEmpty())

{

tmp = head;

head = head->next;

delete tmp;

}

}/\*-------------------------------------------------------------------------\*/

## class Lexeme

class Lexeme

{

protected:

string name;

int type;

public:

Lexeme(string s, int \_type) : type(\_type), name(s){}

Lexeme(char s, int \_type) : type(\_type)

{

name.push\_back(s);

};

int getType() { return type; }

string getName() { return name; }

virtual int getPriority() { return -1; }

virtual double binaryCalculate(Lexeme \*l1, Lexeme \*l2) { return -1; }

virtual double unaryCalculate(Lexeme \*l1) { return -1; }

virtual double getVal() { return -1; }

virtual bool operator > (Lexeme& l){ return false; }

Lexeme& operator=(const Lexeme& l)

{

if (this == &l)

return \*this;

name = l.name;

type = l.type;

return \*this;

}

bool operator==(const Lexeme &l){ return (type == l.type && name == l.name); }

bool operator!=(char s){ return (name[0] != s); }

};

## class LexemeNum: public Lexeme

{

double val;

public:

LexemeNum():Lexeme() {};

LexemeNum(char s, int \_type, double \_val) : Lexeme(s, \_type), val(\_val) {};

LexemeNum(string s, int \_type, double \_val) : Lexeme(s, \_type), val(\_val){ };

double getVal() { return val; }

bool operator==(int num)

{

return ( val == (double)num);

}

};

## class LexemeVar: public Lexeme

public:

static double ar[26];

static bool isVar[26];

static int count;

LexemeVar() :Lexeme() {};

LexemeVar(char s, int \_type) :Lexeme(s, \_type)

{

count++;

isVar[s - 'a'] = true;

};

double getVal()

{

return ar[name[0]-'a'];

}

static void clear()

{

for (int i = 0; i < 26; i++)

{

ar[i] = 1;

isVar[i] = false;

}

}

};

## class LexemeOper: public Lexeme

{

protected:

int priority = lowest;

public:

LexemeOper(char s, int \_type) : Lexeme(s, \_type) { };

int getPriority() { return priority; }

bool operator >= (Lexeme& l)

{

return (getPriority() >= l.getPriority());

}

bool operator > (Lexeme& l)

{

return (getPriority() > l.getPriority());

}

};

## class UnaryOper: public LexemeOper

{

public:

UnaryOper(char s, int \_type) : LexemeOper(s, \_type) { priority = high; }

double unaryCalculate(Lexeme \*l1)

{

switch (name[0])

{

case '-': return (-l1->getVal()); break;

case 'c': return (cos(l1->getVal())); break;

case 's': return (sin(l1->getVal())); break;

default: throw UNKNOWN\_TYPE; break;

}

}

};

## class BinaryOper : public LexemeOper

{

public:

BinaryOper(char s, int \_type) : LexemeOper(s, \_type)

{

if (name == "+" || name == "-") priority = low;

else if (name == "\*" || name == "/" || name == "^") priority = medium;

}

double binaryCalculate(Lexeme \*l1, Lexeme \*l2)

{

switch (name[0])

{

case '+': return (l1->getVal() + l2->getVal()); break;

case '-': return (l1->getVal() - l2->getVal()); break;

case '\*': return (l1->getVal() \* l2->getVal()); break;

case '/': return (l1->getVal() / l2->getVal()); break;

case '^': return (pow(l1->getVal(),l2->getVal())); break;

default: throw UNKNOWN\_TYPE; break;

}

}

};

## Выделение лексем

int NewRead(string &s, List <Lexeme\*>&l, int numLex = 0)

{

int statusRead, status = wait, statusBrackets = 0, length = s.length();

char tmp;

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

{

numLex++;

tmp = s[i];

statusRead = GetReadStatus(&s[i],i, length);

switch (status)

{

case wait://if end - error

switch (statusRead)

{

case num:

//считали число

ReadNum(l, &s[i], i);

status = nextOp;

break;

case var:

//закинули лексему переменную

ReadVar(l, tmp);

status = nextOp;

break;

case leftB:

//считали скобку, число левых скобок++

ReadLeftB(l, statusBrackets);

status = wait;

break;

case minus1:

//считали унарный минус

ReadUnarOp(l, tmp, i);

status = nextEx;

break;

case unOp:

//считали унарный -

ReadUnarOp(l, tmp, i);

status = nextEx;

break;

case funct:

//считали скобку, число левых скобок++

ReadUnarOp(l, tmp, i);

status = wait;

break;

default:throw ERROR\_READ;break;

}

break;

case nextOp:

switch (statusRead)

{

case binOp:

//считали бинарный оп

ReadBinarOp(l, tmp);

status = nextEx;

break;

case minus1:

//считали бинарный -

ReadBinarOp(l, tmp);

status = nextEx;

break;

case binOpPost:

//считали бинарный постфиксный опер

ReadBinarOp(l, tmp);

status = nextOpBin;

break;

case rightB:

//считали правую скобку, число правых скобок--

ReadRightB(l, statusBrackets);

status = nextOp;

break;

default:throw ERROR\_READ;break;

}

break;

case nextEx://if end - error

switch (statusRead)

{

case num:

//считали число

ReadNum(l, &s[i], i);

status = nextOp;

break;

case var:

//считали var

ReadVar(l, tmp);

status = nextOp;

break;

case leftB:

//считали скобку, число левых скобок++

ReadLeftB(l, statusBrackets);

status = wait;

break;

case funct:

ReadUnarOp(l, tmp, i);

status = wait;

break;

default:throw ERROR\_READ; break;

}

break;

case nextOpBin:

switch (statusRead)

{

case binOp:

//считали бинарный оperand

ReadBinarOp(l, tmp);

status = nextEx;

break;

case minus1:

//считали бинарный -

ReadBinarOp(l, tmp);

status = nextEx;

break;

case rightB:

//считали правую скобку, число скобок--

ReadRightB(l, statusBrackets);

status = nextOp;

break;

default:throw ERROR\_READ; break;

}

break;

default:throw ERROR\_READ; break;

}

}

if (status == wait || status == nextEx || statusBrackets != 0)

throw ERROR\_READ;

l.reverce();

return numLex;

}

## Перевод в польскую запись

Stack<Lexeme\*>\* GetPolishNotation(List <Lexeme\*>&list, int size)

{

Stack<Lexeme\*> \*res = new Stack<Lexeme \*>(size);

Stack<Lexeme \*> tmp(size);

Lexeme \*l;

while (!list.isEmpty())

{

l = list.quick\_pop\_front();

switch (l->getType())

{

case number:

{

res->quickPush(l);

} break;

case var:

{

res->quickPush(l);

} break;

case binary\_oper:

{

while (!tmp.isEmty() && \*tmp.showTop() > \*l)

res->quickPush(tmp.quickPop());

tmp.quickPush(l);

} break;

case unary\_oper:

{

while (!tmp.isEmty() && \*tmp.showTop() > \*l)

res->quickPush(tmp.quickPop());

tmp.quickPush(l);

} break;

case left\_brackets:

{

tmp.quickPush(l);

} break;

case right\_brackets:

{

while (\*tmp.showTop() != '(')

res->quickPush(tmp.quickPop());

tmp.quickPop();

} break;

}

}

while (!tmp.isEmty())

res->quickPush(tmp.quickPop());

return res;

}

## Вычисление

double GetAnswer(Stack<Lexeme\*>\* st)

{

Lexeme \*l,\*l1, \*l2, \*count;

Stack <Lexeme\*> tmp(st->getNumTop());

for (int i = 0; i<=st->getNumTop(); i++)

{

l = st->getElem(i);

switch (l->getType())

{

case number: tmp.quickPush(l); break;

case var: tmp.quickPush(l); break;

case binary\_oper:

l1 = tmp.quickPop();

l2 = tmp.quickPop();

count = new LexemeNum('e', number, l->binaryCalculate(l2,l1));

tmp.quickPush( count);

break;

case unary\_oper:

l1 = tmp.quickPop();

count = new LexemeNum('e', number, l->unaryCalculate(l1));

tmp.quickPush(count);

break;

}

}

return tmp.showTop()->getVal();

}

## Сумма ряда

int main()

{

List <Lexeme\*> l;

string s;

double ans = 0;

int numLex, n;

cout << "print stop if you want to exit\n";

while (1)

{

cout << "print n member of functional series\n";

cin >> s;

if (s == "stop")

break;

try

{

numLex = NewRead(s, l);

}

catch (...)

{

cout << "error of print\n";

l.~List();

LexemeVar::clear();

continue;

}

Stack<Lexeme\*> \*pSt = GetPolishNotation(l, numLex);

LexemeVar::blockVar('n');

ReadVar();

cout << "print n ";

cin >> n;

for (int i = 1; i <= n; i++)

{

LexemeVar::setVar('n', i);

ans += GetAnswer(pSt);

}

cout << "Your answer " << ans << "\n\n";

ans = 0;

LexemeVar::clear();

}

return 0;

}