Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

**Институт информационных технологий, математики и механики**

**Отчёт**

«Арифметические выражения»

**Выполнил:**

Дубов Алексей Леонидович

Группа 0823-3

**Проверил:**

Козинов Евгений Александрович

г. Нижний Новгород

2016 г.

Оглавление

[Содержание 2](#_Toc446450755)

[Введение 3](#_Toc446450756)

[Постановка задачи 4](#_Toc446450757)

[Руководство пользователя 5](#_Toc446450758)

[Руководство программиста 6](#_Toc446450759)

[Заключение 8](#_Toc446450761)

[Список Используемой литературы: 9](#_Toc446450762)

[Приложения 10](#_Toc446450763)

# 

# Введение

Для представления арифметических и логических выражений часто используется польская запись, которая просто и точно указывает порядок выполнения операций. Кроме того, она не требует скобок. В этой записи, впервые примененной польским логиком Я. Лукашевичем, операторы следуют непосредственно за операндами. Однако первое теоретическое описание и практическое применение алгоритма польско-инверсной запись предложил и обосновал нидерландский учёный **Э́ дсгер Ви́ бе Де́ йкстра.**

В общем виде польско-инверсная запись выглядит следующим образом:

1. Запись набора операций состоит из последовательности операндов и знаков операций. Операнды и операции в выражении при письменной записи разделяются пробелами.
2. Выражение читается слева направо. Когда в выражении встречается знак операции, то выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Полученный результат заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.
3. Результатом вычисления становится результат последней вычисленной операции.

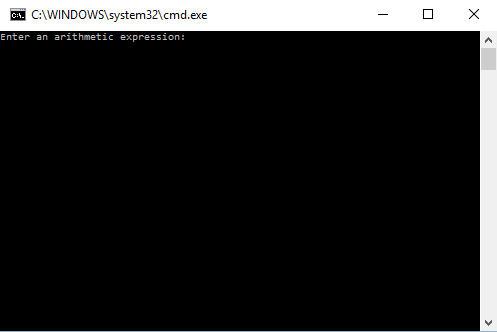
# Постановка задачи

В данной лабораторной работе было необходимо написать программу, реализующую метод перевода арифметических выражений в польско- инверсную запись. Под арифметическим выражением понимается строка, которая состоит из операндов, операций и скобок. Под операцией понимается символ или набор символов, который выполняет какие-то определенные действия над операндами (сложение, вычитание, умножение, деление и возведение в степень). Операндами в данном случае являются числовые константы. Константа – это любое действительное число. Скобки определяют порядок выполнения операций, наряду с приоритетами, которые определены правилами арифметики. В записи арифметического выражения, не содержащего ошибок, количество открывающихся скобок должно быть равно количеству закрывающихся.

Обработка арифметического выражения происходит в два этапа. На первом этапе арифметическое выражение приводиться к виду удобному для вычисления значения выражения компьютером. В данной лабораторной работе используется преобразование в «польско-инверсную запись» или «постфиксную запись». Такое выполняется с использованием динамической структуры – такой как стек На втором этапе, при заданных значениях переменных, происходит вычисление значения арифметического выражения, для этого так же используется стек.

# Руководство пользователя

После запуска программы пользователю предлагается ввести арифметическое выражение. Для начала работы необходимо ввести выражение, например: 1+2-(3\*4+10/5), и нажать Enter.



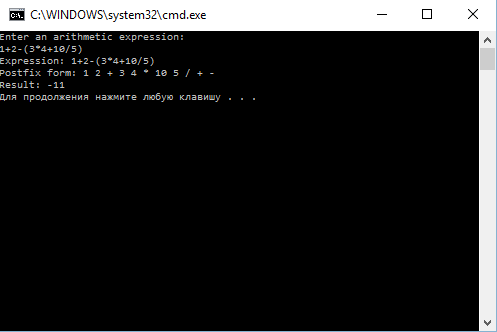
**Рис 1.** Ввод выражения.

После выполнения программы на экран выводится три строки:

Expression: 1+2-(3\*4+10/5) (содержит введенное пользователем выражение.)

Postfix form: 1 2 + 3 4 \* 10 5 / + - (содержит выражение в постфиксной форме записи.)

Result: -11 (содержит результат вычисления выражения)



**Рис 2**. Результат работы программы.

# 

# Руководство программиста

Для реализации алгоритмов было использовано два класса:

* Шаблонный класс «Стек» (Stack).
* Класс создающий польско-инверсную запись и вычисляющий результат (Postfix).

**Описание классов и его основных методов:**

**Шаблонный класс «Стек» (Stack).**

Динамическая структура стек определяется операциями: взять и положить, связанными отношением «LIFO» (последний вошел – первый вышел).

template <typename T>

class Stack

{

private:

T \*array;

const int size;

int top;

public:

Stack(int s=15); // Конструктор инициализации

~Stack(); // Деструктор

void push(const T &val ); // Положить элемент в стек

T pop(); // Взять элемент из стека

const T &peek(); // Показать верхушку стека

int getSize(); //Получить размер стека

int getTop(); //Получить номер текущего элемента

};

**Класс формирующий польско-инверсную запись и вычисляющий результат (Postfix).**

class Postfix

{

public:

string Convert(string input); // перевод инфиксной формы в постфиксную

double Result(string input); // подсчёт результата

int IsOperator(char c); // проверка на операцию

int GetOperationPrt(char c); // определение приоритета операции

};

# Заключение

В результате выполнения данной лабораторной работы было создано консольное приложение, позволяющее вычислять арифметические выражения. Все классы были протестированы по отдельности, а затем вместе на разнообразных примерах.

Таким образом, задачи, поставленные в начале, были успешно выполнены, а результаты проверены.

# Список используемой литературы:

1. Б. Страуструп Язык программирования С++. Специальное издание. Пер. с англ. – М.: ООО «Бином-Пресс», 2008 г. – 1104 с.: ил.
2. Гергель В.П. Рабочие материалы к учебному курсу «Методы программирования», ННГУ, 2002. – 100 c.

**Список используемых Интернет-ресурсов:**

1. <https://ru.wikipedia.org/wiki/Обратная_польская_запись>
2. <http://www.intuit.ru/studies/courses/2193/67/lecture/1980?page=1>

# 

# Приложения

**Класс Stack:**

template <typename T>

class Stack

{

private:

T \*array;

const int size;

int top;

public:

Stack(int s=15); // Конструктор инициализации

~Stack(); // Деструктор

void push(const T &val ); // Положить элемент в стек

T pop(); // Взять элемент из стека

const T &peek(); // Показать верхушку стека

int getSize(); //Получить размер стека

int getTop(); //Получить номер текущего элемента

};

template <typename T>

Stack<T>::Stack(int s):size(s), top(-1)

{

if (size>MaxSize) throw "Size must be <=100";

if (size<0) throw "Size must be >=0";

array = new T[size];

}

template <typename T>

Stack<T>::~Stack()

{

delete [] array;

}

template <typename T>

void Stack<T>::push(const T &val)

{

if (top+1>=size) throw "Stack is full";

array[++top] = val;

}

template <typename T>

T Stack<T>::pop()

{

if (top<0) throw "Stack is empty";

return array[top--];

}

template <class T>

const T &Stack<T>::peek()

{

if (top<0) throw "Stack is empty";

return array[top];

}

template <typename T>

int Stack<T>::getSize()

{

return size;

}

template <typename T>

int Stack<T>::getTop()

{

return top;

}

**Класс Postfix:**

class Postfix

{

public:

string Convert(string input); // перевод инфиксной формы в постфиксную

double Result(string input); // подсчёт результата

int IsOperator(char c); // проверка на операцию

int GetOperationPrt(char c); // определение приоритета операции

};

int Postfix::IsOperator(char c) // проверка на операцию

{

if (c=='+' || c=='-' || c=='\*' || c=='/' || c=='^' || c=='('|| c==')') return 1;

else if (c==' ' || c=='=') return 2;

else return 0;

}

int Postfix::GetOperationPrt(char c) // определение приоритета операции

{

switch (c)

{

case '(': return 0;

case ')': return 1;

case '+': return 2;

case '-': return 2;

case '\*': return 3;

case '/': return 3;

case '^': return 4;

default: return -1;

}

}

string Postfix::Convert(string infix)

{

int len = infix.length();

string polish; // Строка для обратной польской записи

Stack<char> OperationStack(len); // Стек для операторов

bool var = true;

int count1=0,count2=0;

for (int i=0;i<len;i++) // Проверка на правильный ввод

{

if ((!isdigit(infix[i])) && (!IsOperator(infix[i]))) // Проверяем есть ли посторонние символы

throw "Error! Invalid character" ;

if (isdigit(infix[i])) var = false; // Проверяем: выражение не может быть без цифр

if ((infix[i])=='(') count1++; // Проверяем: кол-во '(' и ')' должно быть равно

if ((infix[i])==')') count2++;

if (strchr("+-/\*^", infix[i])!=NULL && strchr("+-/\*^", infix[i+1])!=NULL && i<len-1) // Проверяем: орерации не могут быть друг за другом

throw "Error! Operations are not consistent" ;

}

if (var == true) throw "Error! The expression does not contain numbers" ;

if (count1!=count2) throw "Error! Not agreed number of brackets" ;

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

{

if ((i==1) && (infix[0]=='-')) polish="-"; // Если в начале минус

if ((IsOperator(infix[i]))==2) continue; // Если пробел или равно, пропускаем

if (isdigit(infix[i])) // Если цифра,

{

while (!IsOperator(infix[i])) // считываем всё число

{

polish+=infix[i++];

if (i==len) break;

}

polish.push\_back(' '); // добавляем пробел после числа

i--;

}

if ((IsOperator(infix[i]))==1) // если операция или скобки

if (infix[i] == '(' && infix[i+1] == '-' && i<len-2) // Если '-' после '('

{

i=i+2;

if (isdigit(infix[i]))

{

polish.push\_back('-');

while (!IsOperator(infix[i]))

{

polish+=infix[i++];

if (i==len) break;

}

polish.push\_back(' ');

}

}

else if (infix[i]=='(') OperationStack.push(infix[i]);

else if (infix[i]==')')

{

char s = OperationStack.pop();

while (s != '(')

{

polish.push\_back(s);

polish.push\_back(' ');

s=OperationStack.pop();

}

}

else if (i!=0) // если не минус в начале

{

while ((OperationStack.getTop()>-1) && (GetOperationPrt(infix[i])<=GetOperationPrt(OperationStack.peek())))

{

polish.push\_back(OperationStack.pop());

polish.push\_back(' ');

}

OperationStack.push(infix[i]);

}

}

//Когда прошли по всем символам, выкидываем из стека все оставшиеся там операторы в строку

while (OperationStack.getTop()>-1)

{

polish.push\_back(OperationStack.pop());

polish.push\_back(' ');

}

return polish;

}

double Postfix::Result(string polish) // вычисление результата

{

double res=0;

int len = polish.length();

Stack<double> ResultStack(len);

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

{

//Если цифра, то читаем все число и толкаем на вершину стека

if (isdigit(polish[i]))

{

string str;

double op;

while (!IsOperator(polish[i]))

{

str+=polish[i++];

if (i==len) break;

}

istringstream(str)>>op;

ResultStack.push(op);

i--;

}

// Если встречен минус

if (polish[i]=='-' && isdigit(polish[i+1]) && i<len-1)

{

i=i+1;

string str;

double op;

while (!IsOperator(polish[i]))

{

str+=polish[i++];

if (i==len) break;

}

istringstream(str)>>op;

ResultStack.push(-op);

i--;

}

else if (((IsOperator(polish[i]))==1) && (i!=0))

{

double op1=ResultStack.pop();

double op2=ResultStack.pop();

switch (polish[i])

{

case '+': res = op2 + op1; break;

case '-': res = op2 - op1; break;

case '\*': res = op2 \* op1; break;

case '/':

if (op1!=0)

{

res = op2 / op1; break;

}

else throw "Division by 0!";

case '^': res = pow(op2,op1); break;

}

ResultStack.push(res);

}

}

return ResultStack.peek();

}