МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Отчет по Лабораторной работе**

**Постфиксная нотация**

**Выполнил:** студент группы 381808-2

Пасухин Дмитрий Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2019

Содержание

[1. Введение 3](#_Toc24750272)

[2. Постановка задачи 4](#_Toc24750273)

[3. Руководство пользователя 5](#_Toc24750274)

[4. Руководство программиста 7](#_Toc24750275)

[4.1 Описание структуры программы 7](#_Toc24750276)

[4.2 Описание структур данных 8](#_Toc24750277)

[4.3 Описание алгоритмов 10](#_Toc24750278)

[5. Заключение 11](#_Toc24750279)

[Приложение: Фрагменты исходного кода 12](#_Toc24750280)

[ReversePolish.h 12](#_Toc24750281)

[ReversePolish.cpp 13](#_Toc24750282)

# Введение

В математике существует древняя традиция помещать оператор между операндами (x+y), а не после операндов (xy+). Форма с оператором между операндами называется инфиксной записью. Форма с оператором после операндов называется постфиксной, или обратной польской записью в честь польского логика Я. Лукасевича (1958).

Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул. Во-первых, любая формула может быть выражена без скобок. Во-вторых, она [удобна для вычисления формул в машинах со стеками](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C#.D0.92.D1.8B.D1.87.D0.B8.D1.81.D0.BB.D0.B5.D0.BD.D0.B8.D1.8F_.D0.BD.D0.B0_.D1.81.D1.82.D0.B5.D0.BA.D0.B5). В-третьих, инфиксные операторы имеют приоритеты, которые произвольны и нежелательны. Например, мы знаем, что ab+c значит (ab)+c, а не a(b+c), поскольку произвольно было определено, что умножение имеет приоритет над сложением. Но имеет ли приоритет сдвиг влево перед операцией И? Кто знает? Обратная польская запись позволяет устранить такие недоразумения.

# Постановка задачи

**Цель данной работы** — разработка библиотеки, позволяющей переводить выражения из префиксного вида в постфиксный,способной подсчитать результат. Также освоение таких инструментов разработки программного обеспечения, как система контроля версий Git и фрэймворк для разработки автоматических тестов Google Test.

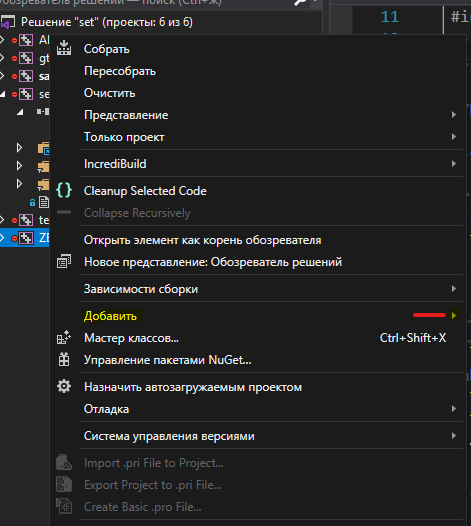
Выполнение работы предполагает решение следующих задач:

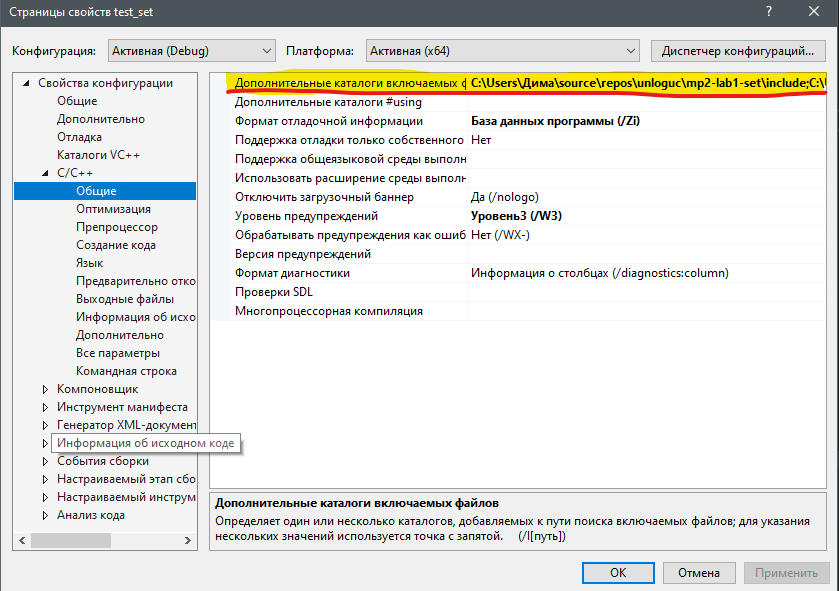
1. Реализация класса *RevPol* согласно заданному интерфейсу.
2. Обеспечение работоспособности примера использования.
3. Публикация исходных кодов в личном репозитории на GitHub.

# Руководство пользователя

В ходе лабораторной работы была получена статическая библиотека(staticlib.lib), для того чтобы ее использовать необходимо:

1. В рабочий проект добавить проект статической библиотеки ReversePolish.vcxproj
2. В свойство вашего проекта указать ссылку на статическую библиотеку



1. Сделать включение заголовочного фала ReversePolish.h, указав в свойствах своего проекта где необходимо искать заголовочный файл
2. В результате можно пользоваться классом ReversePolish и его методами. Например, результат выполнения следующего кода:

#include "ReversePolish.h"

#include <string>

int main()

{

*setlocale*(*LC\_ALL*, "Russian");

string str{ "154+245\*3+4-5+(8+1)" };

RevPol a(str);

a.Print\_Begin();

a.Print\_Revers();

*cout* << a.Result();

*system*("pause>nul");

return 0;

}

Результат:



# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

Модуль *ReversePolish* – статическая библиотека. Содержит заголовочные файлы *ReversePolish.h* в которых определены соответственно интерфейсы классов для обраной польской записи перечисления EType, ставящие в соответствие тип операции, перечисление Status, ставящие в соответствие статус операции, класс Type, хранящий элимент и все перечисления, характеризующие этот элемент. Файлы, содержащие их реализацию – *ReversePolish.cpp*

* Модуль *test\_ ReversePolish*. Содержит для класса *ReversePolish* набор тестов, реализованный в файлах *test\_ ReversePolish.cpp* с помощью использования фреймворка Google Test

Модуль *sample\_prime\_numbers*. Содержит файл *sample\_prime\_numbers.cpp*, в котором реализован пример использования класса *ReversePolish* выводящее результат и запись обратной польской записи выражения.

## Описание структур данных

Класс *Type* реализован путем наследования класса *std::string*

Поля класса:

* *EType type;* хранение типа строки
* *Priority pr;хранение приоритета строки*
* *std::string str; хранение строки*

#### Класс RevPol – класс постфиксной польской записи

**Элементы, объявленные со спецификатором private:**

*std::vector<Type> tmp;* временный вектор, для хранения анализированной строки от пользователя, хранящий объекты типа Type

*Status status;* Поле, хранящее статус класса на данной итерации

*std::string exprs;* Поле, хранящее начальную строку

*std::stack<Type> oper;* Стек, хранящий операции для преобразования в постфиксный вид

*std::vector<std::string> polexprs;* Вектор, хранящий строку в постфиксном виде

*bool is\_operation(char);* Метод для проверки символа является он операцией или нет

*bool is\_digit(const char&);* Метод для проверки символа является он числом или нет

*bool is\_ParTh(const char&);* Метод для проверки символа является он скобочкой или нет

*void reads();* Метод для анализа введённой строки, расставляющий приоритеты и типы

*void calculate();*Метод для преобразования в постфиксную нотацию

*void One(int);* Метод для совершения действия типа: загузить в стек операция

*void Two(int);* Метод для совершения действия типа: остаться на этой же итерации, но выгрузить из стека в строку операцию

*void Three(int);* Метод для совершения действия типа: удалить из стека последний элемент

*void Four(int);* Информирование о завершение алгоритма

*void Five(int);*Информирование о ошибке алгоритма

**Со спецификатором доступа public:**

*RevPol(std::string);* Конструктор, принимающий строку

*void Print\_Revers();* Метод, печатающий обратную польскую запись

*void Print\_Begin();* Метод, печатающий введеную пользователем строку

*double Result();*Метод, возвращающей результат выражения

Дружественные функции:

*friend ostream &operator<<(ostream &ostr, const RevPol&RP)* – вывод битового поля на консоль.

## Описание алгоритмов

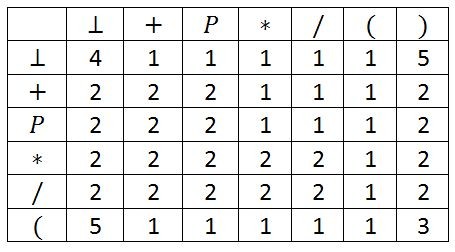
Рассмотрим некоторые алгоритмы, работа которых не очевидна на первый взгляд.

В данной работе были использованы два алгоритма: теория автоматов и алгоритм, созданный Дейкстра.

Теория автоматов

|  |  |
| --- | --- |
| Статус системы | Ожидаемый тип |
| wait | Унарная операция  Открывающая скобочка  Лексема |
| wait\_operation | Бинарная операция  Закрывающая скобочка |
| Wait\_lexeme | Открывающая скобочка  Лексема |

Алгоритм, придуманный Дейкстра:



*One;* загузить в стек операция

*Two;* остаться на этой же итерации, но выгрузить из стека в строку операцию

*Three;* удалить из стека последний элемент

*Four;* Информирование о завершение алгоритма

*Five;*Информирование об ошибке алгоритма

# Заключение

В ходе лабораторной работы был получен класс на основе которого была создана библиотека, позволяющая переводить выражение из префиксной записи в постфиксную и выводить результат выражения.

# Приложение: Фрагменты исходного кода

## ReversePolish.h

#include <iostream>

#include <queue>

#include <vector>

#include <string>

#include <cstdlib>

#include <stack>

enum class Priority

{

NUl=-1,Z=0,O,T,F

};

enum class Status

{

Wait = 0,

Wait\_Lexeme,

Wait\_Operation,

Done,

Error

};

enum class EType

{

Operation = 0,

Lexeme,

ParTh

};

class Type :*std*::string

{

public:

EType type;

Priority pr;

*std*::string str;

Type(*std*::string st, EType t, Priority p=Priority::NUl) :*std*::string(st)

{

type = t;

pr=p;

str = st;

}

};

class RevPol

{

private:

*std*::*vector*<Type> tmp;

Status status;

*std*::string exprs;

*std*::*stack*<Type> oper;

*std*::*vector*<*std*::string> polexprs;

bool is\_operation(char);

bool is\_digit(const char&);

bool is\_ParTh(const char&);

void reads();

void calculate();

void One(int);

void Two(int);

void Three(int);

void Four(int);

void Five(int);

public:

RevPol(*std*::string);

void Print\_Revers();

void Print\_Begin();

double Result();

friend *std*::*ostream*& operator<<(*std*::*ostream*& ostr, const RevPol& RP)

{

<for (const auto& v : polexprs)

{

ostr << v << " ";

}

ostr << *std*::*endl*;

}

};

## ReversePolish.cpp

#include "ReversePolish.h"

#include <cmath>

RevPol::RevPol(*std*::string str)

{

status = Status::Wait;

exprs = str;

calculate();

}

bool RevPol::is\_operation(char in)

{

return (in == '+' || in == '-' || in == '\*' || in == '/' || in == '^');

}

bool RevPol::is\_digit(const char& in)

{

return (in >= '0' && in <= '9');

}

bool RevPol::is\_ParTh(const char& in)

{

return (in == '(' || in == ')');

}

double RevPol::Result()

{

*std*::*vector*<double> tmp;

for (int i=0;i<polexprs.*size*();i++)

{

*std*::string v = polexprs[i];

if (v == "+" || v == "-" || v == "\*" || v == "/" || v == "^")

{

if (tmp.*size*() == 1)

{

if (v == "-")

tmp[0] = -tmp[0];

}

else

{

if (v == "+")

{

tmp[tmp.*size*() - 2] += tmp[tmp.*size*() - 1];

}

else if (v == "-")

{

tmp[tmp.*size*() - 2] -= tmp[tmp.*size*() - 1];

}

else if (v == "\*")

{

tmp[tmp.*size*() - 2] \*= tmp[tmp.*size*() - 1];

}

else if (v == "/")

{

tmp[tmp.*size*() - 2] /= tmp[tmp.*size*() - 1];

}

else if (v == "^")

{

tmp[tmp.*size*() - 2] = *pow*(tmp[tmp.*size*() - 2],tmp[tmp.*size*() - 1]);

}

tmp.*pop\_back*();

}

}

else

{

tmp.*push\_back*(*stod*(v));

}

}

if (tmp.*size*()!=1)

{

*std*::*cout* << "Error" << *std*::*endl*;

}

return tmp.*at*(0);

}

void RevPol::reads()

{

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

{

if (is\_digit(exprs[i]))

{

*std*::string st;

st = exprs[i];

while (is\_digit(exprs[i + 1]))

{

i++;

st += exprs[i];

}

Type temp(st, EType::Lexeme);

tmp.*push\_back*(temp);

}

else if (is\_operation(exprs[i]))

{

*std*::string st;

st = exprs[i];

Type temp(st, EType::Operation);

if ((exprs[i] == '+' || exprs[i] == '-') & (i == 0 || exprs[i - 1] == '('))

temp.pr = Priority::O;

else if ((exprs[i] == '\*' || exprs[i] == '/'|| exprs[i] == '^'))

temp.pr = Priority::T;

else if ((exprs[i] == '+' || exprs[i] == '-'))

temp.pr = Priority::F;

tmp.*push\_back*(temp);

}

else if (is\_ParTh(exprs[i]))

{

*std*::string st;

st = exprs[i];

Type temp(st, EType::ParTh);

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

temp.pr = Priority::Z;

else

temp.pr = Priority::F;

tmp.*push\_back*(temp);

}

else throw "Error input";

}

}

void RevPol::calculate()

{

reads();

for (int i = 0; i < tmp.*size*(); i++)

{

switch (status)

{

case Status::Wait:

switch (tmp[i].type)

{

case EType::Operation:

switch (tmp[i].pr)

{

case Priority::O:

if (tmp[i].str == "-")

{

One(i);

status = Status::Wait\_Lexeme;

}

break;

default:

status = Status::Error;

break;

}

break;

case EType::Lexeme:

polexprs.*push\_back*(tmp[i].str);

status = Status::Wait\_Operation;

break;

case EType::ParTh:

if (tmp[i].pr == Priority::F)

status = Status::Error;

else

{

oper.*push*(tmp[i]);

status = Status::Wait;

}

break;

default:

status = Status::Error;

break;

}

break;

//Done

case Status::Wait\_Lexeme:

switch (tmp[i].type)

{

case EType::Operation:

status = Status::Error;

break;

case EType::Lexeme:

polexprs.*push\_back*(tmp[i].str);

status = Status::Wait\_Operation;

break;

case EType::ParTh:

if (tmp[i].pr == Priority::F)

status = Status::Error;

else

{

oper.*push*(tmp[i]);

status = Status::Wait;

}

break;

default:

status = Status::Error;

break;

}

break;

case Status::Wait\_Operation:

switch (tmp[i].type)

{

case EType::Operation:

if (tmp[i].pr==Priority::F)

{

if (oper.*empty*())

{

One(i);

status = Status::Wait\_Lexeme;

}

else {

switch (oper.*top*().pr)

{

case Priority::T:

//polexprs.push\_back(oper.top().str);

//oper.pop();

Two(i);

i--;

status = Status::Wait\_Operation;

break;

case Priority::F:

//polexprs.push\_back(oper.top().str);

//oper.pop();

Two(i);

i--;

status = Status::Wait\_Operation;;

break;

case Priority::Z:

One(i);

status = Status::Wait\_Lexeme;

break;

case Priority::O:

Two(i);

i--;

status = Status::Wait\_Operation;

break;

default:

status = Status::Error;

break;

}

}

}

else if (tmp[i].pr == Priority::T)

{

if (oper.*empty*())

{

One(i);

status = Status::Wait\_Lexeme;

}

else {

switch (oper.*top*().pr)

{

case Priority::T:

//polexprs.push\_back(oper.top().str);

//oper.pop();

Two(i);

i--;

status = Status::Wait\_Operation;;

break;

case Priority::F:

//oper.push(tmp[i]);

One(i);

status = Status::Wait\_Lexeme;;

break;

case Priority::Z:

One(i);

status = Status::Wait\_Lexeme;

break;

case Priority::O:

One(i);

status = Status::Wait\_Lexeme;

break;

break;

default:

status = Status::Error;

break;

}

}

}

else

{

status = Status::Error;

}

break;

case EType::Lexeme:

status = Status::Error;

break;

case EType::ParTh:

if (tmp[i].pr == Priority::Z)

status = Status::Error;

else

{

if (oper.*empty*() )

status = Status::Error;

else if(oper.*top*().type==EType::Operation)

{

/\*polexprs.push\_back(oper.top().str);

oper.pop();\*/

Two(i);

i--;

status = Status::Wait\_Operation;

}

else if (oper.*top*().type == EType::ParTh&&oper.*top*().pr==Priority::Z)

{

//oper.pop();

Three(i);

status = Status::Wait\_Operation; //наверное

}

else

{

status = Status::Error;

}

}

break;

default:

status = Status::Error;

break;

}

break;

case Status::Done:

i = tmp.*size*() + 222;

break;

case Status::Error:

*std*::*cout* << "Error" << *std*::*endl*;

//throw "Error";

goto MOMO;

break;

default:

Status::Error;

break;

}

}

if (!oper.*empty*() && status != Status::Error)

{

int si = oper.*size*();

for (int j = 0; j < si; j++)

{

polexprs.*push\_back*(oper.*top*().str);

oper.*pop*();

}

}

MOMO:

int k;

//std::cout << "Error" << std::endl;

}

void RevPol::One(int ind)

{

oper.*push*(tmp[ind]);

}

//Cделать после вызова ind--

void RevPol::Two(int ind)

{

polexprs.*push\_back*(oper.*top*().str);

oper.*pop*();

}

void RevPol::Three(int ind)

{

oper.*pop*();

}

//меняет статус на Done

void RevPol::Four(int ind)

{

status = Status::Done;

}

//меняет стутус на Error

void RevPol::Five(int ind)

{

status = Status::Error;

}

void RevPol::Print\_Revers()

{

for (const auto& v:polexprs)

{

*std*::*cout* << v << " ";

}

*std*::*cout* << *std*::*endl*;

}

void RevPol::Print\_Begin()

{

*std*::*cout* << exprs;

*std*::*cout* << *std*::*endl*;

}