Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Нижегородский государственный университет  
им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики (ИИТММ)

Отчёт по лабораторной работе

**Вычисление арифметических выражений**

Выполнил:  
 студент ИИТММ гр. 0823-1  
 Красикова Е.А.

Проверил:

к.т.н., ассистент каф. ПрИнж  
 ИИТММ

Сиднев А.А.

Нижний Новгород  
2016г.

Содержание

[Введение 3](#_Toc470539993)

[Постановка учебно-практической задачи 4](#_Toc470539994)

[Руководство пользователя 5](#_Toc470539995)

[Руководство программиста 6](#_Toc470539996)

[Общее описание структуры программного комплекса. 6](#_Toc470539997)

[Описание структур данных. 6](#_Toc470539998)

[Описание алгоритмов. 7](#_Toc470539999)

[Алгоритм преобразования выражения из инфиксной формы в ОПН 7](#_Toc470540000)

[Алгоритм вычисления выражения, представленного в ОПН 9](#_Toc470540001)

[Заключение 10](#_Toc470540002)

[Список литературы 11](#_Toc470540003)

[Приложение 1. Arithm.h 12](#_Toc470540004)

[Приложение 2. Arithm.cpp 12](#_Toc470540005)

[Приложение 3. StructCNode.h 17](#_Toc470540006)

[Приложение 4. MyStack.h 17](#_Toc470540007)

# Введение

Одной из главных пpичин, лежащих в основе появления языков пpогpаммиpования высокого уpовня, явились вычислительные задачи, требующие больших объёмов рутинных вычислений. Поэтому к языкам пpогpаммиpования предъявлялись требования максимального приближения фоpмы записи вычислений к естественному языку математики. В этой связи одной из первых областей системного пpогpаммиpования сфоpмиpовалось исследование способов выpажений. В данной лабораторной работе будет исследован метод тpансляции с помощью обpатной польской записи, котоpую пpедложил польский математик Я. Лукашевич.[1]

# Постановка учебно-практической задачи

Необходимо разработать класс для разбора и   
вычисления арифметических выражений. Выражение может содержать скобки,   
операции +, -, \*, /, ^ (возведение в степень), || (модуль), константы и   
символьные переменные (строчные буквы латинского алфавита). Необходимо   
реализовать класс, который принимает на вход строку, содержащую выражение,   
выполняет её разбор, выводит сообщение об ошибке при её обнаружении,   
выполняет вычисление значения выражения при заданных значениях переменных.   
Разбор и хранение выражения должны осуществляться в обратной польской   
записи. Реализовать тесты, содержащие различные типы выражений.

# Руководство пользователя

Интерфейс класса для разбора и вычисления арифметических выражений:

class ArithmExp {

char \*exp;

char \*parametr;

int \*pvalue;

int np;

char \*rpn;

bool IsOp(char c);

int Priority(char op);

public:

ArithmExp(char \*exp);

~ArithmExp(void);

void InputParametr(char c, int value);

char\* ReversePolishNotation();

int Calc();

};

Алгоритм работы с классом ArithmExp:

1. Создание объекта (ввод строки-выражения в качестве параметра конструктора-инициализатора).
2. Перевод в обратную польскую запись при помощи метода ReversePolishNotation.
3. Ввод значений параметров в качестве аргументов метода InputParametr.
4. Получение результата при помощи метода Calc.

Пункт 2 можно опустить (в таком случае метод Calc сам вызывает метод ReversePolishNotation).

# Руководство программиста

## Общее описание структуры программного комплекса.

Разработанная программа включает:

1. заголовочный файл «ArithmExp.h» (содержит интерфейс класса ArithmExp)
2. файл исходного кода «ArithmExp.cpp» (содержит реализацию методов класса ArithmExp)
3. заголовочный файл «StructCNode.h» (содержит объявление шаблонной структуры CNode)
4. заголовочный файл «MyStack.h» (содержит интерфейс шаблонного класса MyStack и реализацию его методов)
5. файл исходного кода «test.cpp» (содержит тесты, проверяющие корректность работы методов класса ArithmExp)

## Описание структур данных.

template <class MyType>

struct CNode { // шаблонная структура для хранения звена списка

MyType data; // поле-данные

CNode \*next; // поле-указатель на следующее звено списка

};

template <class ValType>

class MyStack { // шаблонный класс для реализации стека на основе

списка

CNode<ValType> \*top; // указатель на вершину стека

public:

MyStack(void); // конструктор

~MyStack(void); деструктор

ValType& GetTop(); // метод, возвращающий по ссылке элемент,

лежащий на вершине стека

bool pop(); // метод, удаляющий элемент с вершины стека

bool push(ValType x); // метод, добавляющий элемент на вершину

стека

bool IsEmpty(); // Метод, выполняющий проверку стека на пустоту

};

class ArithmExp { // Класс, осуществляющий разбор и вычисление арифметических выражений

char \*exp; // Строка, в которой хранится арифметическое выражение

char \*parametr; // Строка, в которой хранятся имена параметров

int \*pvalue; // Массив значений параметров

int np; // Количество введенных параметров

char \*rpn; // строка, в которой хранится обратная польская запись

bool IsOp(char c); // Метод, определяющий, является ли символ знаком операции

int Priority(char op); // Метод, определяющий приоритет операции

public:

ArithmExp(char \*exp //Указатель на строку-выражение  
 ); // Конструктор-инициализатор

~ArithmExp(void); // Деструктор

void InputParametr(char c, // Имя параметра

int value // Значение параметра

); // Метод, осуществляющий ввод параметра

char\* ReversePolishNotation(); // Метод, преобразующий исходное

выражение в обратную польскую запись (в случае некорректного выражения   
вызывает исключение)

int Calc(); // Метод, выполняющий вычисления (в случае некорректного выражения вызывает исключение)

};

## Описание алгоритмов.

Чтобы вычислить арифметическое выражение, введем промежуточный этап, на котором исходное выражение представим в виде обратной польской нотации. Обратная польская нотация (ОПН) — форма записи [математических](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) и логических выражений, в которой [операнды](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D0%BD%D0%B4) расположены перед знаками [операций](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F_(%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)). Удобство обратной польской нотации заключается в том, что выражения, представленные в такой форме, очень легко вычислять, причём за линейное время. [2]

### Алгоритм преобразования выражения из инфиксной формы в ОПН

В данной лабораторной работе используется алгоритм Дейкстры для преобразования выражений из инфиксной нотации в ОПН. Алгоритм основан на стеке. В преобразовании участвуют две текстовых переменных: входная и выходная строки. В процессе преобразования используется стек, хранящий ещё не добавленные к выходной строке операторы. Преобразующая программа читает входную строку последовательно символ за символом (символ — это не обязательно буква), выполняет на каждом шаге некоторые действия в зависимости от того, какой символ был прочитан.[3]

В программе данный алгоритм дополнен проверкой выражения на корректность. Исключения, свидетельствующие об ошибке, будут вызваны в случаях:

1. Выражение содержит недопустимые символы
2. Количество открывающихся скобок (модулей) и не равно количеству закрывающихся
3. Не введены значения параметров

Начало алгоритма

1. Совершаем обход строки, пока не встречен ноль
2. Проверяем, является ли первый элемент строки минусом. Если да, то делаем вывод, что он унарный, и записываем в выходную строку, после него записываем пробел как некоторое указание на унарный минус
3. Если минус встречен не на первой позиции, проверяем, является ли предыдущий элемент открывающейся скобкой (модулем). Если да, то записываем в выходную строку унарный минус
4. Если текущий элемент не является знаком операции, то
5. Если текущий элемент – строчная латинская буква, то записываем её в выходную строку
6. Если текущий элемент – цифра, то переписываем цифры из исходной строки в выходную пока не встретится знак операции. Если встретился недопустимый символ, выдать сообщение об ошибке
7. В противном случае, символ не является ни знаком операции, ни строчной латинской буквой, ни цифрой, следовательно, это недопустимый элемент и нужно выдать сообщение об ошибке (вызвать исключение)
8. Если стек пуст, или на вершине стека открывающаяся скобка (модуль), или текущий символ – открывающаяся скобка (модуль), то
9. Кладем символ в стек
10. Если это открывающаяся скобка (модуль) увеличиваем соответствующий счетчик на 1
11. Если текущий символ - закрывающаяся скобка, то
12. Уменьшаем счетчик скобок на 1
13. Пока на вершине стека не окажется открывающая скобка, выталкиваем содержимое стека в выходную строку
14. Удаляем открывающуюся скобку из стека
15. Если текущий символ – закрывающийся модуль, то
16. Уменьшаем счетчик модулей на 1
17. Пока на вершине стека не окажется модуль, выталкиваем содержимое стека в выходную строку
18. Удаляем модуль из стека
19. Записываем две черты модуля в выходную строку
20. Если текущий символ – знак возведения в степень, то добавляем его в стек
21. Если текущий элемент - другая операция, то
22. Выталкиваем из стека операции с большим либо равным приоритетом
23. Кладем на вершину стека текущую операцию
24. Если счетчик скобок (модулей) не равен нулю, вызвать исключение
25. Пока стек не пуст, выталкивать содержимое в выходную строку

Конец алгоритма

Обработка операции возведения в степень осуществляется отдельно от других операций ввиду её правоассоциативности.

Чтобы установить, является ли модуль открывающимся, проверялись следующие условия:

1. Счетчик модулей равен 0
2. Предыдущий символ был открывающимся модулем (и был добавлен в стек)
3. Предыдущий символ – арифметическая операция (+, -, \*, /, ^)

### Алгоритм вычисления выражения, представленного в ОПН

Автоматизация вычисления выражений в обратной польской нотации основана на использовании стека.

Начало алгоритма

1. Совершаем обход ОПН, пока не встретится ноль
2. Если текущий символ не знак операции, то
3. Если текущий символ – строчная латинская буква, то
4. проверяем указатель на массив имен параметров: если он равен нулю, выдать сообщение об ошибке и вызвать исключение
5. совершаем обход массива имен параметров: если элемент массива совпал с текущим символом, кладем в стек соответствующее значение параметра из массива значений параметров (элемент с тем же индексом)
6. если не встречено совпадения с элементом массива имен параметров, выдать сообщение об ошибке и вызвать исключение
7. Если текущий символ – цифра, то переводим операнд из символьного представление в число типа int и кладем его в стек
8. Если текущий символ – знак операции, то
9. Достаем из стека нужное количество операндов
10. Выполняем соответствующую операцию над ними
11. Результат кладем на вершину стека
12. Если в стеке остался один элемент, то это результат
13. В противном случае, выражение некорректно и нужно вызвать исключение

Конец алгоритма

Контроль корректности выражения осуществляется путем проверки следующих условий:

1. Указатель на ОПН не равен нулю (метод, преобразующий перевод выражения из инфиксной формы в ОПН не выявил ошибку)
2. В процессе вычислений не встретилась попытка деления на ноль
3. После всех вычислений в стеке остался один элемент

# Заключение

Разработан класс для разбора и вычисления арифметических выражений, использующий алгоритм перевода выражения в обратную польскую нотацию и вычисление выражения в форме ОПН. Класс вычисляет корректные выражения и вызывает исключение в случае недопустимых выражений. Реализованы необходимые тесты.

# Список литературы

1. [http://algolist.manual.ru/syntax/revpn.php] - [1]
2. [http://www.e-maxx-ru.1gb.ru/algo/expressions\_parsing] - [2]
3. Т. Пратт, М. Зелковиц. Языки программирования: разработка и реализация = Terrence W. Pratt, Marvin V. Zelkowitz. Programming Languages: Design and Implementation. — 4-е издание. — Питер, 2002.
4. Бьёрн Страуструп.  Язык программирования C++ = The C++ Programming Language / Пер. с англ. — 3-е изд. — СПб.; М.: Невский диалект — Бином, 1999.

# Приложение 1. Arithm.h

#ifndef C\_\_GIT\_ARITHMETICEXPRESSIONS\_ARITHMETICEXPRESSIONS\_INCLUDE\_ARITHMEXP\_H\_

#define C\_\_GIT\_ARITHMETICEXPRESSIONS\_ARITHMETICEXPRESSIONS\_INCLUDE\_ARITHMEXP\_H\_

class ArithmExp {

char \*exp;

char \*parametr;

int \*pvalue;

int np;

char \*rpn;

bool IsOp(char c);

int Priority(char op);

public:

ArithmExp(char \*exp);

~ArithmExp(void);

void InputParametr(char c, int value);

char\* ReversePolishNotation();

int Calc();

};

#endif // C\_\_GIT\_ARITHMETICEXPRESSIONS\_ARITHMETICEXPRESSIONS\_INCLUDE\_ARITHMEXP\_H\_

# Приложение 2. Arithm.cpp

#include "ArithmExp.h"

#include "MyStack.h"

#include "math.h"

#include <iostream>

ArithmExp::ArithmExp(char \*exp\_) {

int n;

for (n=0; exp\_[n] != 0; n++) {}

exp = new char[n+1];

for (int i=0; i < n+1; i++)

exp[i] = exp\_[i];

parametr = 0;

pvalue = 0;

np = 0;

rpn = 0;

}

ArithmExp::~ArithmExp(void) {

delete []exp;

delete []parametr;

delete []pvalue;

}

void ArithmExp::InputParametr(char c, int value) {

np++;

parametr = (char\*)realloc(parametr, (np+1)\*sizeof(char));

parametr[np-1] = c;

parametr[np] = 0;

pvalue = (int\*)realloc(pvalue, np\*sizeof(int));

pvalue[np-1] = value;

}

bool ArithmExp::IsOp(char c) {

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

return true;

else

return false;

}

int ArithmExp::Priority(char op) {

switch (op) {

case '+':

case '-': return 1; break;

case '\*':

case '/': return 2; break;

case '^':

case '|': return 3; break;

}

}

char\* ArithmExp::ReversePolishNotation() {

MyStack<char> st;

int n, k = 0;

int f = 0, br = 0;

bool flag = 0, fl = 0;

for (n=0; exp[n] != 0; n++) {}

for (int i=0; exp[i] != 0; i++) {

fl = 0;

if ( (i == 0) && (exp[i] == '-') ) {

rpn = (char\*)realloc(rpn, (k+2)\*sizeof(char));

rpn[k++] = '-';

rpn[k++] = ' ';

continue;

}

if (i != 0) {

if ( ( (exp[i-1] == '(') && (exp[i] == '-') ) || ( (exp[i-1] == '|') && (exp[i] == '-') ) ) {

rpn = (char\*)realloc(rpn, (k+2)\*sizeof(char));

rpn[k++] = '-';

rpn[k++] = ' ';

continue;

}

}

if (!IsOp(exp[i])) {

if ( ( (exp[i] > 96) && (exp[i] < 123) ) ) {

rpn = (char\*)realloc(rpn, (k+1)\*sizeof(char));

rpn[k++] = exp[i];

} else {

if ( (exp[i] > 47) && (exp[i] < 58) ) {

for (; ( (!IsOp(exp[i])) && (exp[i] != 0) ); i++) {

if (!( (exp[i] > 47) && (exp[i] < 58) ) )

throw 1;

rpn = (char\*)realloc(rpn, (k+1)\*sizeof(char));

rpn[k++] = exp[i];

}

rpn = (char\*)realloc(rpn, (k+1)\*sizeof(char));

rpn[k++] = ' ';

i--;

} else {

throw 1;

}

}

} else {

if ((st.IsEmpty()) || (exp[i] == '(') || ((st.GetTop() == '(') && (exp[i] != ')')) || ((exp[i] == '|') && (f == 0)) || ((st.GetTop() == '|') && (exp[i] != '|')) || ((exp[i] == '|') && (((exp[i-1] == '|') && (flag == 1)) || ((exp[i-1] == '+') || (exp[i-1] == '-') || (exp[i-1] == '\*') || (exp[i-1] == '/') || (exp[i-1] == '^'))))) {

st.push(exp[i]);

flag = 1;

if (exp[i] == '|')

f++;

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

br++;

} else {

if (exp[i] == ')') {

br--;

while (st.GetTop() != '(') {

rpn = (char\*)realloc(rpn, (k+1)\*sizeof(char));

rpn[k++] = st.GetTop();

st.pop();

}

st.pop();

} else {

if ( (exp[i] == '|') && (f > 0) ) {

flag = 0;

while (st.GetTop() != '|') {

rpn = (char\*)realloc(rpn, (k+1)\*sizeof(char));

rpn[k++] = st.GetTop();

st.pop();

}

st.pop();

rpn = (char\*)realloc(rpn, (k+2)\*sizeof(char));

rpn[k++] = '|';

rpn[k++] = '|';

f--;

} else {

if (exp[i] == '^') {

st.push(exp[i]);

} else {

if (Priority(st.GetTop()) < Priority(exp[i])) {

st.push(exp[i]);

} else {

rpn = (char\*)realloc(rpn, (k+1)\*sizeof(char));

rpn[k++] = st.GetTop();

st.pop();

st.push(exp[i]);

}

}

}

}

}

}

}

if ((br != 0) || (f != 0))

throw 1;

while (!st.IsEmpty()) {

rpn = (char\*)realloc(rpn, (k+1)\*sizeof(char));

rpn[k++] = st.GetTop();

st.pop();

}

rpn[k] = 0;

return rpn;

}

int ArithmExp::Calc() {

if (rpn == 0)

ReversePolishNotation();

MyStack<int> st;

int n;

bool fl = 0;

int res;

if (rpn == 0)

throw 1;

for (int i=0; rpn[i] != 0; i++) {

if (!IsOp(rpn[i])) {

if ( ( (rpn[i] > 96) && (rpn[i] < 123) ) ) {

if (parametr == 0) {

std::cout << "Input parametr" << std::endl;

throw 1;

}

for (int j=0; j < np; j++) {

if (rpn[i] == parametr[j]) {

fl = 1;

st.push(pvalue[j]);

}

}

if (fl == 0) {

std::cout << "Input parametr" << std::endl;

throw 1;

}

fl = 0;

continue;

}

int operand = 0;

for (n=0; rpn[i+n] != ' '; n++) {}

for (int j=0; rpn[i] != ' '; i++, j++)

operand+=((rpn[i]-'0')\*pow(10.0, n-1-j));

st.push(operand);

} else {

if (rpn[i] == '^') {

int r;

double l;

if (!st.IsEmpty()) {

r = st.GetTop();

st.pop();

} else {

throw 1;

}

if (!st.IsEmpty()) {

l = st.GetTop();

st.pop();

} else {

throw 1;

}

st.push(pow(l, r));

}

if (rpn[i] == '+') {

int r, l;

if (!st.IsEmpty()) {

r = st.GetTop();

st.pop();

} else {

throw 1;

}

if (!st.IsEmpty()) {

l = st.GetTop();

st.pop();

} else {

throw 1;

}

st.push(l+r);

}

if (rpn[i] == '-') {

if (rpn[i+1] == ' ') {

i+=2;

double operand = 0;

for (n=0; rpn[i+n] != ' '; n++) {}

for (int j=0; rpn[i] != ' '; i++, j++)

operand+=((rpn[i]-'0')\*pow(10.0, n-1-j));

st.push(-operand);

} else {

int r, l;

if (!st.IsEmpty()) {

r = st.GetTop();

st.pop();

} else {

throw 1;

}

if (!st.IsEmpty()) {

l = st.GetTop();

st.pop();

} else {

throw 1;

}

st.push(l-r);

}

}

if (rpn[i] == '\*') {

int r, l;

if (!st.IsEmpty()) {

r = st.GetTop();

st.pop();

} else {

throw 1;

}

if (!st.IsEmpty()) {

l = st.GetTop();

st.pop();

} else {

throw 1;

}

st.push(l\*r);

}

if (rpn[i] == '/') {

int r, l;

if (!st.IsEmpty()) {

r = st.GetTop();

st.pop();

} else {

throw 1;

}

if (!st.IsEmpty()) {

l = st.GetTop();

st.pop();

} else {

throw 1;

}

if (r == 0)

throw 1;

else

st.push(l/r);

}

if (rpn[i] == '|') {

int tmp;

if (!st.IsEmpty()) {

tmp = st.GetTop();

st.pop();

} else {

throw 1;

}

st.push(abs(tmp));

i++;

}

}

}

res = st.GetTop();

st.pop();

if (st.IsEmpty()) {

return res;

} else {

std::cout << "error expression";

throw 1;

}

}

# Приложение 3. StructCNode.h

#ifndef C\_\_GIT\_PALINDROM\_IMPLEMENTATION\_PALINDROM\_IMPLEMENTATION\_INCLUDE\_STRUCTCNODE\_H\_

#define C\_\_GIT\_PALINDROM\_IMPLEMENTATION\_PALINDROM\_IMPLEMENTATION\_INCLUDE\_STRUCTCNODE\_H\_

#include "stdlib.h"

template <class MyType>

struct CNode {

MyType data;

CNode \*next;

};

#endif // C\_\_GIT\_PALINDROM\_IMPLEMENTATION\_PALINDROM\_IMPLEMENTATION\_INCLUDE\_STRUCTCNODE\_H\_

# Приложение 4. MyStack.h

#ifndef C\_\_GIT\_PALINDROM\_IMPLEMENTATION\_PALINDROM\_IMPLEMENTATION\_INCLUDE\_MYSTACK\_H\_

#define C\_\_GIT\_PALINDROM\_IMPLEMENTATION\_PALINDROM\_IMPLEMENTATION\_INCLUDE\_MYSTACK\_H\_

#include "StructCNode.h"

template <class ValType>

class MyStack {

CNode<ValType> \*top;

public:

MyStack(void);

~MyStack(void);

ValType& GetTop();

bool pop();

bool push(ValType x);

bool IsEmpty();

};

template <class ValType>

MyStack<ValType> :: MyStack(void) {

top = 0;

}

template <class ValType>

MyStack<ValType> :: ~MyStack(void) {

while ( top != 0 ) {

CNode<ValType> \*tmp = top;

top = top->next;

delete tmp;

}

}

template <class ValType>

ValType& MyStack<ValType> :: GetTop() {

if ( top != 0 )

return top->data;

else

throw 1;

}

template <class ValType>

bool MyStack<ValType> :: push(ValType x) {

CNode<ValType> \*node = new CNode<ValType>;

node->data = x;

node->next = top;

top = node;

return true;

}

template <class ValType>

bool MyStack<ValType> :: pop() {

if ( top != 0 ) {

CNode<ValType> \*tmp = top;

top = top->next;

delete tmp;

return true;

} else {

return false;

}

}

template <class ValType>

bool MyStack<ValType> :: IsEmpty() {

if (top == 0)

return true;

else

return false;

}

#endif // C\_\_GIT\_PALINDROM\_IMPLEMENTATION\_PALINDROM\_IMPLEMENTATION\_INCLUDE\_MYSTACK\_H\_