Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

**Разбор и вычисление арифметических выражений**

Выполнила:

студентка ИИТММ гр. 813-2

Усова М.А.

Проверил:

ассистент каф. ПИнж, ИИТММ

Сиднев А.А.

Нижний Новгород

2016 г.

Содержание

[Введение 3](#_Toc470037905)

[Постановка задачи 4](#_Toc470037906)

[Руководство пользователя 5](#_Toc470037907)

[Руководство программиста 6](#_Toc470037908)

[Описание структуры программы 6](#_Toc470037909)

[Описание структур данных 7](#_Toc470037910)

[Описание алгоритмов 7](#_Toc470037911)

[Заключение 10](#_Toc470037912)

[Литература 11](#_Toc470037913)

[Приложение 12](#_Toc470037914)

[Пример работы с классом 12](#_Toc470037915)

[Partition 13](#_Toc470037916)

[Convert to polish 14](#_Toc470037917)

[Calculator 16](#_Toc470037918)

# Введение

Данный тип лабораторной работы направлен на изучение обратной польской записи. Обратная польская запись совершенно унифицирована — она принципиально одинаково записывает унарные, бинарные, тернарные и любые другие операции, а также обращения к функциям, что позволяет не усложнять конструкцию вычислительных устройств при расширении набора поддерживаемых операций.

Помимо знакомства с обратной польской записью, в процессе написания работы студент должен использовать изученные средства языка С++, уметь работать со списком и стеком. Кроме этого, лабораторная работа подразумевает тестирование программы с помощью Google Tests.

# Постановка задачи

Необходимо реализовать 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 необходимо осуществлять в обратной польской записи. Необходимо реализовать тесты, содержащие различные типы выражений (не менее 15 тестов).

# Руководство пользователя

Для вычисления арифметических выражений пользователю доступен класс Polish:

class Polish {

Variable\* vars; // переменные

int priority(char); // приоритет операций

CNode\* partition(char\*); // разбиение выражения на части

bool IsTrueNumber(char); // проверка корректности ввода

bool IsTrueSymbol(char);

public:

void AddVar(char, int); // добавляет (изменяет) переменную

char\* ConvertToPolish(char\*); // преобразование в польскую запись

double Calculator(char\*); // вычисляет выражение

};

Для того чтобы вычисления значения арифметического выражения с помощью данного класса необходимо[[1]](#footnote-1):

1. Создать объект класса Polish:

Polish Expression;

1. Ввести арифметическое выражение:

expression = “(1+2\*a+|-3|+4)/4”;

1. Воспользоваться функцией для перевода выражения в обратную польскую запись:

char\* a = Expression.ConvertToPolish(expression);

1. Если выражение содержит переменные, то нужно их добавить:

Expression.AddVar('a', 2);

1. Для вычисления значения выражения, используя обратную польскую запись, используем метод Calculator:

double res = Expression.Calculator(a);

Если было введено выражение с ошибкой или недопустимое значение переменной, пользователь получит сообщение об ошибке.

# Руководство программиста

## Описание структуры программы

Лабораторная работа состоит из следующих проектов:

1. code.lib *–* библиотека, содержащая интерфейс класса Polish и реализацию его методов. Содержит заголовочный файл PolishRecord.h, заголовочный файл Stack.h, содержащий реализацию, файл PolishRecord.cpp .
2. gtest.lib *–* библиотека, содержащая заголовочные файлы, а также реализации, необходимые для работы Google Tests, при которой происходит тестирование программы.
3. mytest.exe *–* приложение, тестирующее корректность работы разработанного класса. Содержит файл test.cpp.

* Файл PolishRecord.cpp содержит реализацию методов класса Polish:
  1. Определение приоритета операций:

int priority(char);

Функция принимает на вход символ и возвращает приоритет этой операции.

* 1. Разбиение выражения на части:

CNode\* partition(char\*);

Функция, разбивающая выражение на части, основанная на методе конечного автомата.

* 1. Проверка корректности символов и чисел:

bool IsTrueNumber(char);

Функция получает на вход символ и, если он является цифрой или переменной, возвращает true, в противном случае – false.

bool IsTrueSymbol(char);

Функция принимает на вход символ и в случае, когда символ есть знак операции +, -, \*, /, ^, |, пробел ” ” или скобки (), возвращает true. В противном случае - false.

* 1. Добавление переменных:

void AddVar(char, int);

Функция, которая добавляет или изменяет переменные, которые хранятся в самом классе.

* 1. Преобразование в обратную польскую запись:

char\* ConvertToPolish(char\*);

Функция принимает на вход строку, а возвращает строку, содержащую выражение в обратной польской записи.

* 1. Вычисление арифметического выражения:

double Calculator(char\*);

Функция принимает на вход строку, содержащую выражение в обратной польской записи, а возвращает значение арифметического выражения.

* В test.cpp содержатся:

1. Тесты, необходимые для проверки корректности методов класса.
2. Функция для запусков всех тестов:

int main(int ac, char\* av[]);

* Файл Stack.h содержит реализацию стека.

## Описание структур данных

В разработанной программе используются следующие структуры данных:

1. Класс для работы с арифметическими выражениями Polish:

class Polish

1. Стек с данными типа <T>, содержащийся в файле Stack.h:

class Stack {

TNode<T>\* top;

public:

Stack() {

top = 0;

}

~Stack();

T& gettop();

bool pop();

void push(T);

bool empty() {

return !top;

}

1. Список с данными типа <T>, содержащийся в файле Stack.h:

struct TNode {

T data;

TNode\* next;

};

1. Список с данными типа char\*, содержащийся в файле PolishRecord.h:

struct CNode {

char\* data;

CNode \*next;

};

1. Список для хранения переменных содержащийся в файле PolishRecord.h:

struct Variable {

char ch;

double val;

Variable\* next;

};

## Описание алгоритмов

1. ***Конечный автомат.[[2]](#footnote-2)***

Разработанная программа осуществляет перевод выражения в польскую запись.   
Предварительно она разбивает выражение на лексемы (метод CNode\* partition(char\*)), используя конечный автомат.

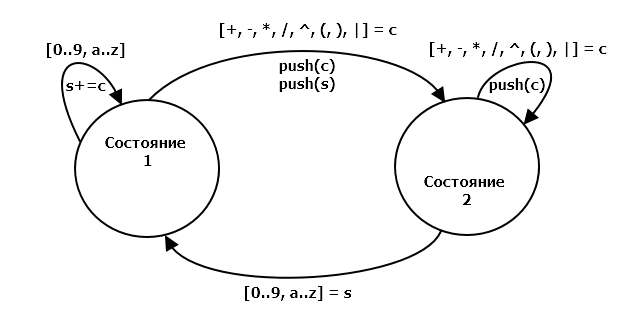
На каждой итерации автомата мы просматриваем i-ый символ строки. Он имеет два положения.

1-ое положение:

* если символ – цифра или буква, то добавляем его значение к предварительно созданной строке;
* если символ допустим для алфавита (+, -, \*, /, ^, |), добавляем его в список лексем и переводим автомат во второе положение.

2-ое положение:

* если символ – цифра или буква, то создаём строку, записываем в неё значение символа и переводим автомат в первое положение;
* иначе – добавляем его в список лексем.



1. ***Обратная польская запись.[[3]](#footnote-3)***

Проходим по списку лексем и смотрим на значение каждой лексемы.

1. Если это число или переменная, то сразу записываем его в польскую запись.
2. Если же это операция, то смотрим приоритет:
   1. если её приоритет больше, чем приоритет лежащей в стеке сверху, она просто кладётся в стек;
   2. если её приоритет меньше, операции из стека изымаются и записываются в польскую запись, пока приоритет не станет больше;
3. Если лексема – скобка или модульная скобка, то открывающаяся скобка кладётся поверх стека и становится новым “дном” стека, пока не встретится закрывающая скобка;
4. Если мы встречаем закрывающую скобку, содержимое стека изымается до открывающейся скобки (сама скобка изымается, но не записывается в польскую запись);
5. Если мы встречаем «-», то мы проверяем, является ли он унарным, т.е. вычислить его местоположение: унарный минус может стоять либо перед открывающей скобкой, либо в самом начале строки. В любом другом случае этот минус либо бинарный, либо ошибочный. Добавляем вместо него «\_» в стек.
6. ***Вычисление арифметического выражения.[[4]](#footnote-4)***

Пока в стеке не осталось элементов, в цикле выполняются следующие операции:

1. Если очередной элемент строки не является знаком арифметической операции, то он заносится в стек;
2. Если же элемент – знак арифметической операции, то из стека изымаются 2 или 1 последние элемента и над ними (ним) производится арифметическая операция.

После того, как дошли до конца строки, при правильно введённом выражении в стеке останется одно число – результат исходного арифметического выражения.

# Заключение

В результате лабораторной работы был реализован класс для разбора и вычисления арифметических выражений. Класс принимает на вход строку, содержащую выражение, выполняет её разбор, выводит сообщение об ошибке при её обнаружении, выполняет вычисление значения выражения при заданных значениях переменных.

Реализованы тесты, содержащие различные типы выражений.

# Литература

1. *Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн К.*. Алгоритмы: построение и анализ. — 2-е изд. — М.: Вильямс, 2005. — С.1296.
2. *Ахо А. В., Хопкрофт Д. Э., Ульман Д. Д..* Структуры данных и алгоритмы. — М.: Вильямс, 2000. — С. 231.
3. Конечный автомат — [https://ru.wikipedia.org/wiki/Конечный\_автомат]
4. Обратная польская запись — [https://ru.wikipedia.org/wiki/Обратная\_польская\_запись]
5. Работа с logic\_error — [http://www.cplusplus.com/forum/beginner/107516/]

# Приложения

## Пример работы с классом

#include "PolishRecord.h"

#include <string.h>

#include <stdio.h>

#include <stdexcept>

#include <clocale>

#include <iostream>

int main() {

using namespace std;

Polish Expression;

char expression[100];

double res;

int var, v, j = 0, p = 0, ch = 0;

int f = 1, stop = 0;

char\* current;

setlocale(LC\_ALL, "Russian");

while (f != 0) {

printf("Выберете вид выражения, с которым будем работать:\n1) численное выражение;\n2) выражение с переменными;\n0) выход из программы.\n");

scanf("%d", &v);

switch (v) {

case 1:

printf("Введите выражение:\n");

scanf("%s", expression);

try

{

current = Expression.ConvertToPolish(expression);

res = Expression.Calculator(current);

printf("Значение выражения равно %f\n", res);

}

catch (std::logic\_error &e) {

cout << e.what() << endl;

f = 1;

stop = 1;

}

break;

case 2:

j = 0;

printf("Введите выражение (переменные a,b,c, ...):\n");

scanf("%s", expression);

try

{

current = Expression.ConvertToPolish(expression);

printf("Введите число переменных.\n");

scanf("%d", &j);

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

printf("Введите значение переменной %c = ", ('a' + i));

scanf("%d", &ch);

Expression.AddVar(('a' + i), ch);

}

res = Expression.Calculator(current);

printf("Значение выражения равно %f\n", res);

}

catch (std::logic\_error &e) {

cout << e.what() << endl;

f = 1;

stop = 1;

}

break;

case 0:

f = 0;

stop = 1;

break;

default:

printf("Ошибка ввода. Нет такого пункта меню. До свидания.\n");

f = 0;

stop = 1;

}

if (stop == 0) {

printf("Далее:\n1) показать польскую запись;\n2) ввести новое выражение;\n0) выход из программы.\n");

scanf("%d", &var);

switch (var) {

case 1:

printf("Польская запись: ");

for (int i = 0; i < strlen(current); ++i)

printf("%c", current[i]);

printf("\n");

break;

case 2:

f = 1;

break;

case 0:

f = 0;

break;

default:

printf("Ошибка ввода. Нет такого пункта меню. До свидания.\n");

f = 0;

}

}

stop = 0;

}

}

## Partition

CNode\* Polish::partition(char\* expression) {

bool status;

CNode\* partition = new CNode;

CNode\* cur = partition;

char \* str = new char[10];

int len = 0, halfpart = 0;

char\* c;

if (!expression[0])

throw std::logic\_error("String is empty\n");

int slen = strlen(expression);

char \*exp = new char[slen + 2];

exp = strcpy(exp, expression);

exp[slen] = ' ';

exp[slen + 1] = 0;

if (IsTrueNumber(exp[0]))

status = 1;

else

status = 0;

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

c = new char(exp[i]);

if (status) {

if (IsTrueNumber(\*c) || ((\*c == '.') && (!halfpart))) {

if (\*c == '.')

halfpart = 1;

str[len] = \*c;

len++;

delete c;

} else {

if (!IsTrueSymbol(\*c))

throw std::logic\_error("Wrong expression\n");

str[len] = 0;

cur = Add(cur, str);

status = 0;

str = new char[10];

len = 0;

halfpart = 0;

if (\*c != ' ')

cur = Add(cur, c);

continue;

}

}

if (!status) {

if (IsTrueNumber(\*c)) {

str[len] = \*c;

len++;

status = 1;

delete c;

} else {

if (!IsTrueSymbol(\*c))

throw std::logic\_error("Wrong expression\n");

if (\*c != ' ') {

cur = Add(cur, c);

}

}

}

}

delete[]exp;

partition = partition->next;

return partition;

}

## Convert to polish

char\* Polish::ConvertToPolish(char\* exp) {

vars = 0;

char\* result = new char[strlen(exp) \* 2];

char last\_element = ' ';

int pos = 0, mod = 0, bracket\_was = 0, module\_was = 0;

CNode\* parsed = partition(exp);

Stack<char> stack;

if (parsed == 0)

throw std::logic\_error("Wrong expression\n");

while (parsed) {

char\* element = parsed->data;

if (IsTrueNumber(\*element)) {

for (int i = 0; element[i] != 0; i++, pos++)

result[pos] = element[i];

result[pos] = ' ';

pos++;

} else {

switch (\*element) {

case '(':

stack.push(\*element);

bracket\_was = 1;

break;

case '|':

if (!pos || mod || (IsTrueSymbol(last\_element)

&& (last\_element != '|'))) {

stack.push(\*element);

if (parsed->next)

if (\*parsed->next->data == '|')

mod = 1;

else

mod = 0;

module\_was = 1;

} else {

while (stack.gettop() != '|') {

result[pos] = stack.gettop();

pos++;

result[pos] = ' ';

pos++;

stack.pop();

}

result[pos] = '|';

pos++;

result[pos] = ' ';

pos++;

stack.pop();

module\_was = 0;

}

break;

case ')':

if (stack.gettop() == '(')

throw std::logic\_error("Wrong expression\n");

if (bracket\_was) {

while (stack.gettop() != '(') {

result[pos] = stack.gettop();

pos++;

result[pos] = ' ';

pos++;

stack.pop();

}

}

else

throw std::logic\_error("Wrong expression\n");

stack.pop();

break;

default:

while (!stack.empty()) {

if ((priority(\*element)) <= priority(stack.gettop())) {

result[pos] = stack.gettop();

pos++;

result[pos] = ' ';

pos++;

stack.pop();

}

else

break;

}

if (\*element == '-') {

if (!pos) {

stack.push('\_');

} else {

if ((last\_element == '|') && (module\_was))

stack.push('\_');

else

if (!stack.empty()) {

if (last\_element == '(')

stack.push('\_');

else

stack.push(\*element);

}

else

stack.push(\*element);

}

}

else

stack.push(\*element);

}

}

parsed = parsed->next;

last\_element = \*element;

}

while (!stack.empty()) {

if ((stack.gettop() != '(') && (stack.gettop() != '|')) {

result[pos] = stack.gettop();

pos++;

result[pos] = ' ';

pos++;

}

else

throw std::logic\_error("Wrong expression\n");

stack.pop();

}

result[pos] = 0;

return result;

}

## Calculator

double Polish::Calculator(char\* PolRec) {

int k = 0;

if (!PolRec)

throw std::logic\_error("Wrong expression\n");

int len = strlen(PolRec), numlen;

Variable\* buf;

Stack<double> stack;

char\* num;

double a, b;

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

if ((PolRec[i] >= 'a') && (PolRec[i] <= 'z')) {

buf = vars;

while (buf) {

if (buf->ch == PolRec[i])

break;

buf = buf->next;

}

if (!buf)

throw std::logic\_error("Variables are not declared\n");

stack.push(buf->val);

i++;

}

else

if ((PolRec[i] >= '0') && (PolRec[i] <= '9')) {

num = new char[10];

numlen = 0;

while (PolRec[i] != ' ') {

if ((PolRec[i] >= 'a') && (PolRec[i] <= 'z'))

throw std::logic\_error("Wrong expression\n");

num[numlen] = PolRec[i];

numlen++;

i++;

}

num[numlen] = 0;

double p = 0.0;

int e = 0;

int c;

while (((c = \*num++) != 0) && (c != '.')) {

p = p\*10.0 + (c - '0');

}

if (c == '.') {

while ((c = \*num++) != 0) {

p = p\*10.0 + (c - '0');

e = e - 1;

}

}

while (e < 0) {

p = p/10.0;

e++;

}

stack.push(p);

}

else

if (!stack.empty()) {

b = stack.gettop();

stack.pop();

if ((PolRec[i] != '|') && (PolRec[i] != '\_')) {

if (!stack.empty()) {

a = stack.gettop();

stack.pop();

}

else

throw std::logic\_error("Wrong expression\n");

}

switch (PolRec[i]) {

case '+':

stack.push(a + b);

break;

case '-':

stack.push(a - b);

break;

case '\*':

stack.push(a \* b);

break;

case '/':

stack.push(a / b);

break;

case '^':

stack.push(pow(a, b));

break;

case '|':

stack.push(abs(b));

break;

case '\_':

stack.push(-b);

break;

default:

throw std::logic\_error("Wrong expression\n");

}

i++;

}

else

throw std::logic\_error("Wrong expression\n");

}

double result = stack.gettop();

stack.pop();

if (stack.empty())

return result;

else

throw std::logic\_error("Wrong expression\n");

}

1. Пример программы работы с классом Polish смотри в приложении «Пример работы с классом» [↑](#footnote-ref-1)
2. См. приложение «Partition» [↑](#footnote-ref-2)
3. См. приложение «Convert to polish» [↑](#footnote-ref-3)
4. См. приложение «Calculator» [↑](#footnote-ref-4)