Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

Отчёт по лабораторной работе №1

**Разбор и вычисление арифметических выражений**

Выполнил:

студент ИТММ гр. 381908-3

Бражник Д.А.

Проверил:

программист ИТММ каф. МОСТ лаб. СТВВ

Усова М.А.

Нижний Новгород

2020 г.

Содержание

[Введение 3](#_Toc23683040)

[Постановка задачи 4](#_Toc23683041)

[Руководство пользователя 5](#_Toc23683042)

[Руководство программиста 6](#_Toc23683043)

[Описание структуры программы 6](#_Toc23683044)

[Описание функций 6](#_Toc23683045)

[Описание структур данных 6](#_Toc23683046)

[Описание алгоритмов 6](#_Toc23683047)

[Заключение 9](#_Toc23683051)

[Литература 10](#_Toc23683052)

[Приложение 11](#_Toc23683053)

[Приложение 1. Текст программы на языке C++ 11](#_Toc23683054)

# Введение

Данная лабораторная работа несёт ценный опыт для меня, ведь разбор и вычисление выражений – это довольно важная часть программирования, любой калькулятор использует данные алгоритмы.

Разбор выражений показывает наглядно, как компьютер может работать с данными типами выражений.

Разбор выражений – это базовые навыки стандартных умений, которые должен уметь реализовывать каждый программист. Более сложные задачи будут требовать реализации более сложных алгоритмов.

# Постановка задачи

# Цель: реализовать класс для разбора и вычисления арифметических выражений.

# Требования к арифметическому выражению

# может содержать скобки (), [], {}, скобки, не поддерживаемые программой должны вызывать исключения;

# может содержать константы и символьные переменные (строчные буквы латинского алфавита);

# поддерживаемые операции: сложение ( + ), вычитание ( ̶̶ ), умножение ( \* ), деление ( / ), возведение в степень ( ^ ), операция взятия модуля ( | | );

# Требования к программе

# Необходимо реализовать класс, который

# принимает на вход строку, содержащую арифметическое выражение;

# выполняет её разбор, выводит сообщение об ошибке в случае некорректного задания выражения;

# выполняет вычисление значения выражения при заданных значениях переменных и выводит результат.

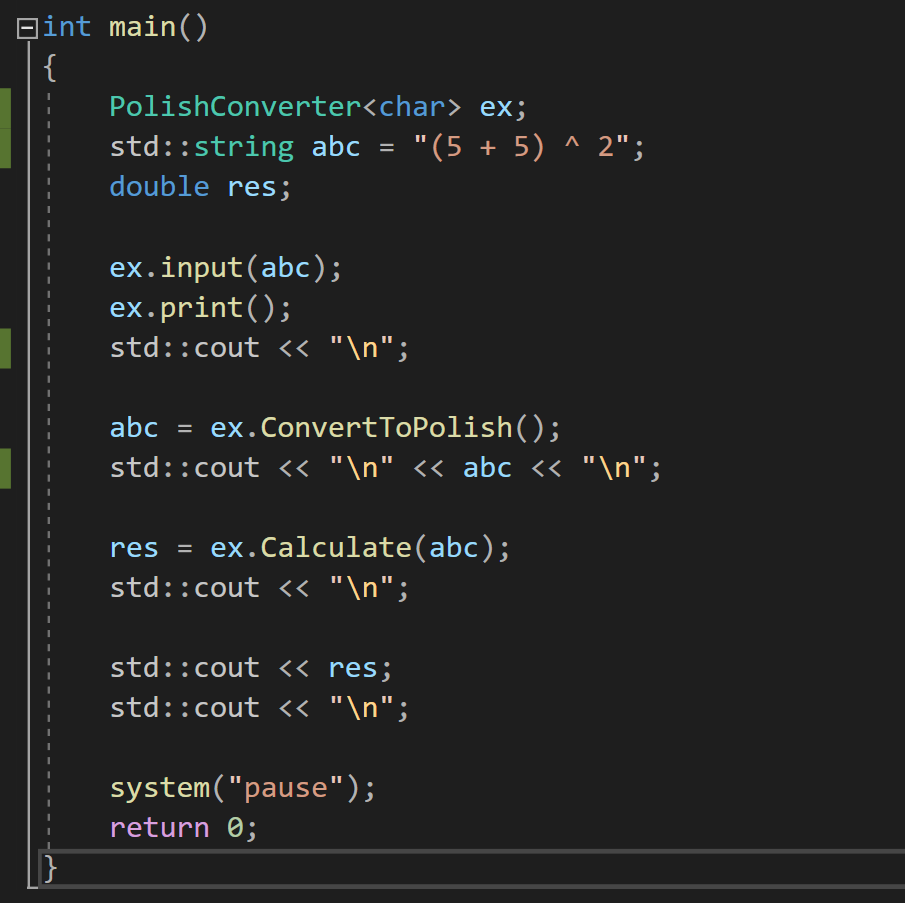
# Разбор и хранение выражения необходимо осуществлять в обратной польской записи.

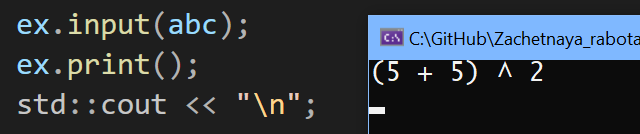
# Необходимо реализовать тесты, содержащие различные типы выражений.

# Руководство пользователя

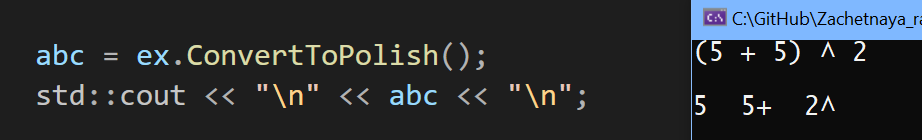
Данная программа написана с помощью программы Microsoft Visual Studio 2019 на языке C++.

Создаём обьект класса, строку которая будет хранить наше выражение, переменную типа double для хранения ответа.

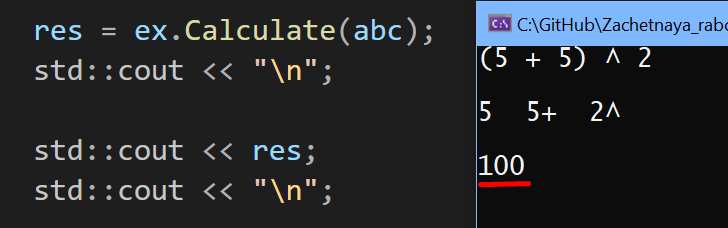


Введём выражение и выведем в консоль чему равен стек.  
 

Преобразуем в обратную польскую запись и посмотрим результат.



Вызовем функцию расчёта и выведем ответ в консоль.



# 

# Руководство программиста

Данная программа реализует следующий набор алгоритмов:

* Разбиение выражения на лексемы;
* Перевод выражения в обратную польскую запись;
* Вычисления значения выражения;
* Проверка правильности ввода выражения;
* Заполнение стека выражением;

## Описание структуры программы

В ходе работы программы на вход ей приходит выражение, которое “кладётся” в стек на массиве.

Для вычисления выражения:

* 1. Вводится арифметическое выражение
  2. Добавляются переменные (если нужно);
  3. Вычисляется польская запись;
  4. Вычисляется выражение.

## Описание функций

int priorit(char ch)- На вход подаётся символ из выражения, на выходе его приоритет.

int correctInput(char ch) - На вход подаётся символ из выражения, на выходе определяется, что это был за символ (цифра,буква,скобка, матем. символ, модуль, точка) относительно возвращенного значения.

double excute\_calc(double k1, double k2, char pst) – На вход поступает 2 числа и матем. символ, внутри функции находятся switch с привязкой к каждому символу, если символ “+”, он сложит k1 и k2 и вернёт значение.

void input(std::string inf) – Процедура для ввода выражения в стек, с проверкой на правильность.

void AddVar(char ch, int value) – Процедура для обозначение переменной в выражении, на вход поступает буква и её значение.

std::string ConvertToPolish(std::string inf) – Функция преобразующая математическое выражение в обратную польскую запись, на входе выражение, на выходе польская запись

std::string ConvertToPolish() – Функция преобразующая заранее введённое выражение и хранящиеся в стеке в обратную польскую запись, возвращает строку с обратной польской записью.

double Calculate(std::string pst) – Функция принимающая на вход строку содержащую обратную польскую запись и возвращающая конечный ответ.

double StrToDouble(std::string str) – Функция преобразующая строку в число с плавающей точкой, точность 15 знаков после запятой и возвращающая его.

int StrToInt(std::string str) - Функция преобразующая строку в целое число и возвращающая его.

void print() – Функция для просмотра введённого выражения в классическом виде.

## Описание структур данных

stack<char> expression – стек на массиве для хранения введённого выражения.

## Описание алгоритмов

**Обратная польская запись**

На вход поступает строка с выражением

1. Берём символ из выражения
2. Проверяем, что это символ (буква, цифра…)
3. Если цифра или буква записываем в массив
4. Если математическая операция, запись в строку
5. Если математическая операция выше по “важности”, записать после 2 чисел
6. Выполнять пока не кончится выражение
7. В конец дописать операции, попавшие в строку (сохраняя порядок)

Пример:

Вход: 3 + 4 \* 2 / (1 - 5)^2

Читаем «3»

Добавим «3» к выходной строке

Выход: 3

Читаем «+»

Кладём «+» в стек

Выход: 3

Стек: +

Читаем «4»

Добавим «4» к выходной строке

Выход: 3 4

Стек: +

Читаем «\*»

Кладём «\*» в стек

Выход: 3 4

Стек: + \*

Читаем «2»

Добавим «2» к выходной строке

Выход: 3 4 2

Стек: + \*

Читаем «/»

Выталкиваем «\*» из стека в выходную строку, кладём «/» в стек

Выход: 3 4 2 \*

Стек: + /

Читаем «(»

Кладём «(» в стек

Выход: 3 4 2 \*

Стек: + / (

Читаем «1»

Добавим «1» к выходной строке

Выход: 3 4 2 \* 1

Стек: + / (

Читаем «−»

Кладём «−» в стек

Выход: 3 4 2 \* 1

Стек: + / ( −

Читаем «5»

Добавим «5» к выходной строке

Выход: 3 4 2 \* 1 5

Стек: + / ( -

Читаем «)»

Выталкиваем «−» из стека в выходную строку, выталкиваем «(»

Выход: 3 4 2 \* 1 5 −

Стек: + /

Читаем «^»

Кладём «^» в стек

Выход: 3 4 2 \* 1 5 −

Стек: + / ^

Читаем «2»

Добавим «2» к выходной строке

Выход: 3 4 2 \* 1 5 − 2

Стек: + / ^

Конец выражения

Выталкиваем все элементы из стека в строку

Выход: 3 4 2 \* 1 5 − 2 ^ / +

#### Вычисление выражения

Алгоритм работы:

1. Нашлось число – записать в стек;
2. Нашлось второе число – записать в стек;
3. Нашёлся математический оператор, посчитать первое и второе число и положить в стек
4. Выполнять пока в стеке не останется последнее число, которое и будет являться ответом

Пример:

Выражение изначально - 9 + 5 + 3 - 2 ^ 4

После обратной польской записи - 9 5 3 + 2 4 ^ - +

Вычисление по шагам:

1. 9 5 3 + 2 4 ^ - +
2. 9 8 2 4 ^ - +
3. 9 8 2 4 ^ - +
4. 9 8 16 - +
5. 9 -8 +
6. 1

Пункт номер 6 будет являться ответом.

# 

# Заключение

На основе данной работы можно сделать выводы, вычисление арифметических выражений – это важная тема в программирование, которую ни в коем случае нельзя пропускать мимо себя.

Мы научились работать с выражениями записанные строкой, преобразовывать выражения в обратную польскую запись и находить её решение.

# Литература

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., ШтайнК.. 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. Википедия Обратная польская запись

[https://ru.wikipedia.org/wiki/Обратная\_польская\_запись#Простой\_пример]

# Приложение

### Приложение 1. Текст программы на языке C++

Класс:

template<typename T>

class PolishConverter

{

public:

PolishConverter();

~PolishConverter();

int priorit(char ch);

int correctInput(char ch);

double excute\_calc(double k1, double k2, char pst);

void input(std::string inf);

void AddVar(char ch, int value);

std::string ConvertToPolish(std::string inf);

std::string ConvertToPolish();

double Calculate(std::string pst);

double StrToDouble(std::string str);

int StrToInt(std::string str);

void print();

private:

stack<char> expression;

};

template<typename T>

inline int PolishConverter<T>::correctInput(char ch)

{

int key = ch;

if ((key > 64 && key < 91) || (key > 96 && key < 123) ||

key == 94) {

return 0;

} // Встречен Заглавные/прописные/степень

if (key > 41 && key < 48 && key != 46) { return 10; } // Оператор

if (key > 47 && key < 58) { return 11; } //Цифра

switch (key)

{

// Круглые

case 40: return 1;

case 41: return 2;

// Фигурные

case 123: return 3;

case 125: return 4;

// Квадратные

case 91: return 5;

case 93: return 6;

// Пробел

case 32: return 7;

// Модуль

case 124: return -2;

// Точка

case 46: return 8;

default:

return -1;

break;

}

}

template<typename T>

inline int PolishConverter<T>::priorit(char ch)

{

switch (ch)

{

case '(': return 0;

case '{': return 0;

case '[': return 0;

case ')': return 1;

case '}': return 1;

case ']': return 1;

case '+': return 2;

case '-': return 2;

case '\*': return 3;

case '/': return 3;

case '^': return 4;

default: return -1;

}

}

template<typename T>

inline void PolishConverter<T>::input(std::string inf)

{

std::string tmp;

std::string copyInf;

copyInf = inf;

int unar;

int brackets[3] = { 0 };

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

{

char ch = copyInf[i];

int checkSym = correctInput(ch);

int nextInd = correctInput(copyInf[i + 1]);

if (checkSym == 1 && nextInd == 10)

{

brackets[0]++;

tmp.append(std::string(1, ch));

unar = copyInf[i + 2] + '0';

tmp.append(std::string(1, '1'));

tmp.append(std::string(1, ' '));

tmp.append(std::string(1, '/'));

tmp.append(std::string(1, ' '));

tmp.append(std::string(1, copyInf[i + 2]));

i += 2;

continue;

}

switch (checkSym)

{

case -1:

throw std::logic\_error("Input error: uncorect symbol");

break;

case 0:

break;

case 10:

break;

case 11:

break;

case 1:

brackets[0]++;

break;

case 2:

brackets[0]--;

break;

case 3:

brackets[1]++;

break;

case 4:

brackets[1]--;

break;

case 5:

brackets[2]++;

break;

case 6:

brackets[2]--;

break;

case 7:

break;

case 8:

break;

case -2:

if (nextInd == 11 || nextInd == 7)

{

continue;

}

if(nextInd == 10)

{

i++;

continue;

}

break;

default:

throw std::logic\_error("Input error: uncorect symbol");

break;

}

tmp.append(std::string(1, ch));

}

if (brackets[0] != 0 || brackets[1] != 0 || brackets[2] != 0) { throw std::logic\_error("Input error: incorrectly placed brackets"); }

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

{

expression.push(tmp[i]);

}

}

template<typename T>

inline void PolishConverter<T>::AddVar(char ch, int value)

{

char temp;

char num = value + '0';

bool isFindVal = false;

stack<char> copy;

while (!expression.empty()) {

copy.push(expression.top());

expression.pop();

}

while (!copy.empty())

{

if (copy.top() == ch)

{

isFindVal = true;

expression.push(num);

}else{ expression.push(copy.top()); }

temp = copy.top();

copy.pop();

}

if (isFindVal == false) { throw std::logic\_error("Logic error: value is not found in expression"); }

}

template<typename T>

inline std::string PolishConverter<T>::ConvertToPolish(std::string inf)

{

stack<char> stackHelper;

std::string res = "";

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

{

char ch = inf[i];

int k = priorit(ch);

if (k == -1)

res.append(std::string(1, ch));

else

if (stackHelper.empty() || k == 0 || k > priorit(stackHelper.top()))

stackHelper.push(ch);

else

{

if (ch == ')' || ch == '}' || ch == ']')

while (true)

{

char sym = stackHelper.top();

stackHelper.pop();

if (sym != '(' && sym != '{' && sym != '[')

res.append(std::string(1, sym));

else

break;

}

else

{

while (!stackHelper.empty())

{

char lastStackEl = stackHelper.top();

stackHelper.pop();

if (priorit(lastStackEl) >= k)

res.append(std::string(1, lastStackEl));

}

stackHelper.push(ch);

}

}

}

while (!stackHelper.empty())

{

char lastStackEl = stackHelper.top();

stackHelper.pop();

if(lastStackEl == '(' || lastStackEl == ')' || lastStackEl == '{' || lastStackEl == '}' || lastStackEl == '[' || lastStackEl == ']') { continue; }

res.append(std::string(1, lastStackEl));

}

return res;

}

template<typename T>

inline double PolishConverter<T>::Calculate(std::string pst)

{

stack<double> stack2;

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

{

std::string correctNum;

char ch = pst[i];

char longNum[10] = "";

int priority = correctInput(ch);

int nextPriority = correctInput(pst[i + 1]);

if (priority == 7) { continue; }

else {

if (priority == 11 && nextPriority == 11)

{

bool isDouble = false;

int j = i;

int p = 0;

while (correctInput(pst[j]) != 10 || correctInput(pst[j]) != 7)

{

if (correctInput(pst[j] == 8)) { isDouble = true; }

if (correctInput(pst[j]) == 10) { break; }

if (correctInput(pst[j]) == 7) { break; }

longNum[p] = pst[j];

p++; j++;

}

for (int g = 0; g < p; g++) { correctNum += longNum[g]; }

i += p - 1;

if(isDouble){ stack2.push(StrToDouble(correctNum)); }

else{ stack2.push(StrToInt(correctNum)); }

}

else {

if (priority == 11 && nextPriority != 8)

{

stack2.push(ch - 48);

}

else {

if (priority == 11 && nextPriority == 8)

{

int j = i;

int p = 0;

while (correctInput(pst[j]) != 10 || correctInput(pst[j]) != 7)

{

if (correctInput(pst[j]) == 10) { break; }

if (correctInput(pst[j]) == 7) { break; }

longNum[p] = pst[j];

p++; j++;

}

for (int g = 0; g < p; g++) { correctNum += longNum[g]; }

i += p - 1;

stack2.push(StrToDouble(correctNum));

}

else

{

double k1 = stack2.top();

stack2.pop();

double k2 = stack2.top();

stack2.pop();

double res = excute\_calc(k2, k1, ch);

stack2.push(res);

}

}

}

}

}

return stack2.top();

}

template<typename T>

inline double PolishConverter<T>::StrToDouble(std::string str)

{

double res = std::stod(str);

return res;

}

template<typename T>

inline int PolishConverter<T>::StrToInt(std::string str)

{

return stoi(str);

}

template<typename T>

inline double PolishConverter<T>::excute\_calc(double k1, double k2, char pst)

{

switch (pst)

{

case '+': return k1 + k2;

case '-': return k1 - k2;

case '\*': return k1 \* k2;

case '/': return k1 / k2;

case '^': return pow(k1,k2);

default: return -1;

}

}