Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

Отчёт по лабораторной работе

**Разбор и вычисление арифметических выражений**

Выполнил:

студент ИТММ гр. 381908-3

Новожилов А.Ю.

Проверил:

Усова М.А.

Нижний Новгород

2020 г.

Содержание

[Введение 3](#_Toc59453889)

[Постановка задачи 4](#_Toc59453890)

[Руководство пользователя 5](#_Toc59453891)

[Руководство программиста 6](#_Toc59453892)

[Описание структуры программы 6](#_Toc59453893)

[Описание методов 6](#_Toc59453894)

[Описание структур данных 8](#_Toc59453895)

[Описание алгоритмов 8](#_Toc59453896)

[Заключение 11](#_Toc59453897)

[Литература 12](#_Toc59453898)

[Приложение 13](#_Toc59453899)

[Приложение 1. Текст программы алгоритмов 13](#_Toc59453900)

# Введение

Вне всяких сомнений, арифметические выражения поистине вездесущи. Алгоритмы их трансляции просты, по-своему изящны и универсальны. Знакомство с ними может сослужить добрую службу любому программисту.

Мне, как будущему программисту, будет очень полезно разобраться в структуре алгоритмов вычисления арифметических выражений и отработать на практике их написание. Овладев данными навыками, я смогу решать более сложные задачи, базирующиеся на этих стандартных умениях.

# Постановка задачи

Цель: реализовать класс для разбора и вычисления арифметических выражений.

Требования к арифметическому выражению:

* может содержать скобки (), по желанию добавить поддержку разного вида скобок [], {}, 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:

1. Принимает на вход строку, содержащую арифметическое выражение;
2. Выполняет её разбор, выводит сообщение об ошибке в случае некорректного задания выражения;
3. Выполняет вычисление значения выражения при заданных значениях переменных и выводит результат.

# Руководство пользователя

Данная программа написана с помощью программы MicrosoftVisualStudio 2019 на языке C++.

Класс предоставляет пользователю 3 метода:

1. std::string ConvertToPolish(std::string \_expression), который принимает на вход строку выражения, переводит выражение в обратную польскую запись и возвращает строку выражения в этой записи.
2. void AddVar(std::string str\_var, double var\_value); С помощью этого метода можно

установить значения в переменные выражения.

1. double Calculate(); Метод считает значение выражения в обратной польской записи и возвращает

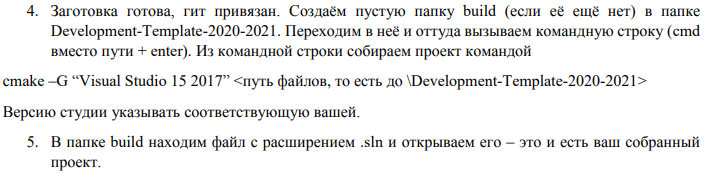


Рисунок 1 Сборка проекта с помощью cmake

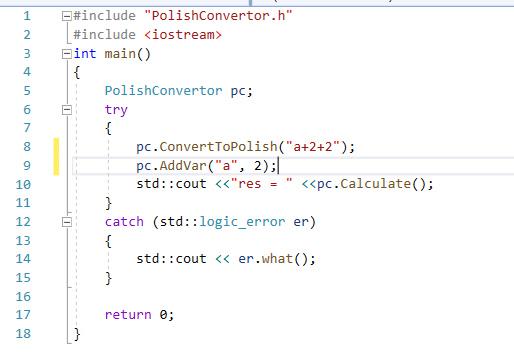


Рисунок 2 Пример использования класса

# Руководство программиста

Данная программа реализует следующий набор алгоритмов:

* Перевод в обратную польскую запись;
* Вычисление выражения, записанного в обратной польской записи;
* Сортировка Хоара.

И следующий набор опций:

* Проверка вводимых данных;
* Проверка последовательности скобок;
* Проверка последовательности операндов и операций.

## Описание структуры программы

Программа состоит из следующих классов:

* PolishConvertor – основной класс;
* Stack – стек на основе списка;
* TList – список;
* TNode –звено списка;

Также были написаны тесты для класса PolishConvertor

## Описание методов

* bool IsCharCorrect(const char& symb); - функция проверки символа. Если символ является допустимым для арифметического выражения, то функция возвращает true, иначе false;
* bool AreParenthesisCorrect(); - функция для проверки последовательности скобок арифметического выражения. Если последовательность скобок является допустимой для арифметического выражения, то функция возвращает true, иначе false;
* bool IsExpressionCorrect(); - функция для проверки корректности всего арифметического выражения;
* void DelSpaces(); - функция удаляет пробелы во входной строке;
* void CreateListOfTerms(); Функция разбирает строковое выражение на элементарные составляющие и записывает в список;
* bool IsSpecialSymb(const char& symb); - служебная функция для проверки, является ли символ символом операции или скобкой;
* void SetUnOperators(); - функция проходит по списку с элементарными составляющими выражения, распознает и устанавливает унарные знаки;
* void CheckSequenceOfTerms(); - функция проверяет корректность последовательности элементарных составляющих выражения;
* void SetValues(); - функция, которая переводит все строковые значения операндов в числовые значения;
* void SetValue(Term& term); - функция разбирает строковое значение и устанавливает числовое значение операнду;
* bool CanSetValue(Term& term)const; - функция, проверяющая можно ли установить числовое значение;
* bool IsDoubleOrPoint(const char ch)const; - функции проверки символа на то, является он символом или точкой.

## Описание структур данных

std::string expression; поле для хранения строкового значения выражения

TList<Term> terms; список для хранения выражения в виде элементарных составляющих

TList<Term> polish\_expression; список для хранения выражения в виде элементарных составляющих в обратной польской записи

int num\_of\_var; переменная для хранения количества переменных в выражении

## Описание алгоритмов[[1]](#footnote-1)

#### Перевод в обратную польскую запись

Рассматриваем поочередно каждый символ:

* Если этот символ - число (или переменная), то просто помещаем его в выходную строку.
* Если символ - знак операции (+, -, \*, / ), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет (допустим он равен 3). Операции сложения и вычитания имеют меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.

Получив один из этих символов, мы должны проверить стек:

1. Если стек все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.
2. Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).

Если текущий символ - открывающая скобка, то помещаем ее в стек.  
Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

Рассмотрим алгоритм на примере простейшего выражения:

Дано выражение: a + ( b - c ) \* d

Рассмотрим поочередно все символы:

Таблица 1 Пример работы алгоритма

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Действие | Состояние выходной строки после совершенного действия | Состояние стека после совершенного действия |
| a | 'a' - переменная. Помещаем ее в выходную строку | a | пуст |
| + | '+' - знак операции. Помещаем его в стек (поскольку стек пуст, приоритеты можно не проверять) | a | + |
| ( | '(' - открывающая скобка. Помещаем в стек. | a | + ( |
| b | 'b' - переменная. Помещаем ее в выходную строку | a b | + ( |
| - | '-' - знак операции, который имеет приоритет 2. Проверяем стек: на вершине находится символ '(', приоритет которого равен 1. Следовательно мы должны просто поместить текущий символ '-' в стек. | a b | + ( - |
| c | 'c' - переменная. Помещаем ее в выходную строку | a b c | + ( - |
| ) | ')' - закрывающая скобка. Извлекаем из стека в выходную строку все символы, пока не встретим открывающую скобку. Затем уничтожаем обе скобки. | a b c - | + |
| \* | '\*' - знак операции, который имеет приоритет 3. Проверяем стек: на вершине находится символ '+', приоритет которого равен 2, т.е. меньший, чем приоритет текущего символа '\*'. Следовательно мы должны просто поместить текущий символ '\*' в стек. | a b c - | + \* |
| d | 'd' - переменная. Помещаем ее в выходную строку | a b c - d | + \* |

#### Алгоритм подсчета выражения в обратной польской записи

Обратная польская запись идеально подходит для вычисления формул на компьютере со стеком. Формула состоит из n символов, каждый из которых является либо операндом, либо оператором. Алгоритм для вычисления формулы в обратной польской записи с использованием стека прост. Нужно просто прочитать обратную польскую запись слева направо. Если встречается операнд, его нужно пометить в стек. Если встречается оператор, нужно выполнить заданную им операцию.

В качестве примера рассмотрим вычисление следующего выражения:

Соответствующая формула в обратной польской записи выглядит так:

Число на вершине стека – это правый операнд (а не левый). Это очень важно для операций деления, вычитания и возведения в степень, поскольку порядок следования операндов в данном случае имеет значение (в отличие от операций сложения и умножения). Другими словами, операция деления действует следующим образом: сначала в стек помещается числитель, потом знаменатель, и тогда операция даёт правильный результат. Отметим, что преобразовать обратную польскую запись в машинный код очень легко: нужно просто двигаться по формуле в обратной польской записи, записывая по одной команде для каждого символа. Если символ является константой или переменной, нужно вписывать команду помещения этой константы или переменной в стек, если символ является оператором, нужно вписывать команду выполнения это операции.

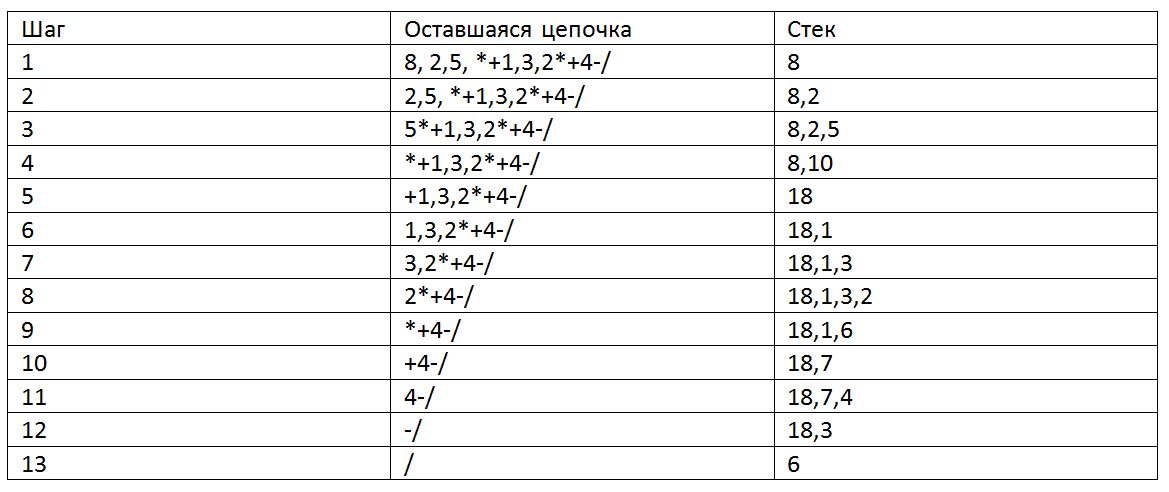


Рисунок 3 Пример работы алгоритма

# Заключение

В ходе выполнения лабораторной работы я изучил алгоритмы разбора и подсчета арифметических выражений. Разработал и протестировал класс, который позволяет проверять и вычислять строковые арифметические выражения.

В итоге, я получил ценные знания об алгоритмах перевода строк в обратную польскую запись, подсчета выражений в обратной польской записи, а также умения применять их на практике, что, несомненно, поможет мне решать более сложные задачи.

# 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. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск. — 2-е изд. — М.: Вильямс, 2001. — С.800.

# Приложение

### Приложение 1. Текст программы алгоритмов

std::string PolishConvertor::ConvertToPolish(std::string \_expression)

{

expression = \_expression;

polish\_expression.DelList();

terms.DelList();

IsExpressionCorrect();

num\_of\_var = 0;

stack<Term> stack;

for (int i = 1; i < terms.GetListLenght() - 1; i++)

{

Term current = terms.get(i);

if (current.type == operand)

{

polish\_expression.InsLast(current);

}

else if (current.type == bin\_operator || current.type == un\_operator)

{

if (stack.empty() || stack.top().prioryty < current.prioryty)

{

stack.push(current);

}

else

{

while (!stack.empty() && stack.top().prioryty >= current.prioryty)

{

polish\_expression.InsLast(stack.top());

stack.pop();

}

stack.push(current);

}

}

else if (current.type == open\_parenthesis)

{

stack.push(current);

}

else if (current.type == close\_parenthesis)

{

while (stack.top().type != open\_parenthesis)

{

polish\_expression.InsLast(stack.top());

stack.pop();

}

stack.pop();

if (current.symbol == "|")

{

stack.push(current);

}

}

}

while (!stack.empty())

{

polish\_expression.InsLast(stack.top());

stack.pop();

}

std::string str\_polish\_expression = "";

for (int i = 0; i < polish\_expression.GetListLenght(); i++)

{

str\_polish\_expression += polish\_expression.get(i).symbol;

str\_polish\_expression += " ";

}

SetValues();

return str\_polish\_expression;

}

double PolishConvertor::Calculate()

{

if (num\_of\_var != 0)

{

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

}

stack<Term> stack;

for (int i = 0; i < polish\_expression.GetListLenght(); i++)

{

Term current = polish\_expression.get(i);

if (current.type == operand)

{

stack.push(current);

}

else if(current.type == bin\_operator)

{

Term operand1 = stack.top();

stack.pop();

double val1 = operand1.value;

Term operand2 = stack.top();

stack.pop();

double val2 = operand2.value;

Term result;

switch (current.symbol[0])

{

case '+':

result.value = val2 + val1;

stack.push(result);

break;

case '-':

result.value = val2 - val1;

stack.push(result);

break;

case '\*':

result.value = val2 \* val1;

stack.push(result);

break;

case '/':

result.value = val2 / val1;

stack.push(result);

break;

case '^':

result.value = pow(val2, val1);

stack.push(result);

break;

}

}

else {

Term operand = stack.top();

stack.pop();

switch (current.symbol[0])

{

case '-':

operand.value \*= -1;

stack.push(operand);

break;

case '|':

if (operand.value < 0)

{

operand.value \*= -1;

}

stack.push(operand);

break;

}

}

}

return stack.top().value;

}

1. Программную реализацию см. в Приложении [↑](#footnote-ref-1)