Федеральное государственное автономное образовательное учреждение  
высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

Отчёт по лабораторной работе

**Преобразование в польскую форму и вычисление арифметических выражений.**

**Выполнил:**

студент ИТММ гр. 0826-1

Поляков М.О.

**Проверил:**

ассистент каф. МОСТ ф-та ИТММ

Кустиков В.Д.

Нижний Новгород

2015 г.

Оглавление

[Постановка задачи 3](#_Toc438582909)

[Руководство пользователя 4](#_Toc438582910)

[Руководство программиста 5](#_Toc438582911)

[Описание структуры программы 5](#_Toc438582912)

[Описание структур данных 6](#_Toc438582913)

[Описание алгоритмов 6](#_Toc438582914)

[Заключение 7](#_Toc438582915)

[Литература 8](#_Toc438582916)

[Приложения 9](#_Toc438582917)

# Постановка задачи

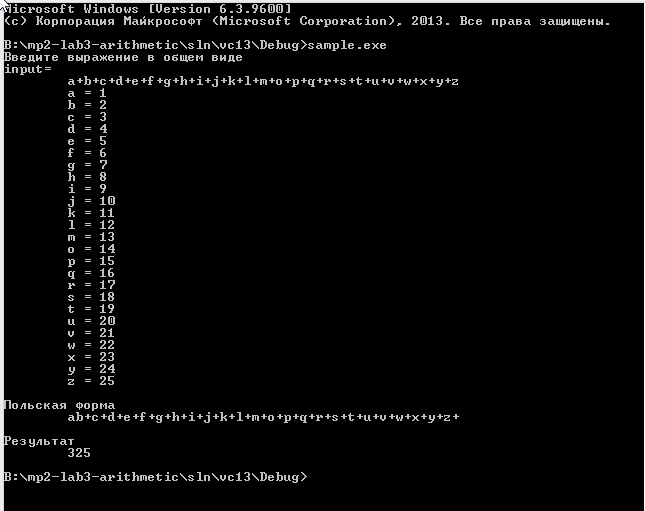
Разработать программу, выполняющую вычисление арифметического выражения с вещественными числами и преобразования данного выражения в общем виде в польскую форму записи. Выражение в качестве операндов может содержать переменные и вещественные числа. Допустимые операции известны: +, -, /, \*. Допускается наличие знака "-" в начале выражения или после открывающей скобки. Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю вид ошибки и номер позиции строки, в которых были найдены ошибки. Все функции должны тестироваться с помощью Google test-ов.

# Руководство пользователя

Пользователю предлагается ввести выражение, в котором допускаются алгебраические операции (+, -, /, \*), открывающие и закрывающие скобки, буквенные символы, которые обозначают переменные.

Если пользователь допустил ошибку в выражении, то программа уведомит его об этом.

Если пользователь ввёл корректные данные, то программа продолжит вычисления. Будет предложено заполнить переменные значениями для выполнения вычислений.



# Руководство программиста

## Описание структуры программы

Ядром проекта является библиотека polish\_notation, схему которой вы видите на Рисунке 1. Стрелка в схеме означает, что выбранный класс использует функционал класса, или функцию по направлению стрелки.

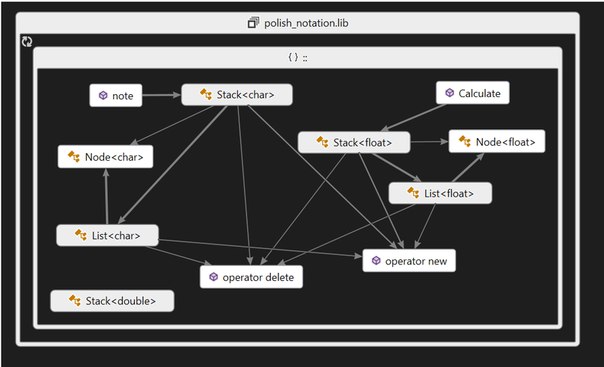


Рисунок 1. Схема зависимостей классов polish\_notation

## Описание структур данных

Входные данные хранятся в виде строки (string). В результате обработки Parser-ом мы получаем массив лексем.

Corrector хранит все найденные ошибки в стеке ошибок (Stack<Error>), который потом выводится в консоль.

* **calculate**

Объявления – calculate.h

Реализация – calculate.cpp

Содержит функцию note и функции Calculate, note

* **stack**

Объявление, реализация – stack.h

Представляет собой обыкновенный стек. Реализован при помощи шаблонов.

Приватные поля:

\* Реализованы два конструктора: пустой и с размером стека

Обладаем методами Put (извлекает и удаляет элемент с верхушки стека), Peek (показывает верхний элемент без удаления), Push (добавляет элемент в верхушку стека). Проверка на переполнение/пустоту осуществляется методами IsFull/IsEmpty.

* **list**

Объявление, реализация –list.h

Представляет собой обыкновенный стек. Реализован при помощи шаблонов.

Приватные поля:

\* Реализованы два конструктора: пустой и с ссылкой на список

Обладаем методами Search (поиск элемента в списке), erase (удаление элемента из списка), InsertFirst (добавляет элемент в начало списка), InsertLast (добавляет элемент в конец списка), InsertAfter (добавляет элемент после какого-либо элемента списка), InsertBefore (добавляет элемент перед каким-либо элементом списка). Возврат указателя на первый элемент getFirst.

## Описание алгоритмов

Рассмотрим пример: (b+a) \* c

Первая лексема – открывающая скобка, она отправляется в стэк. Далее идёт переменная “б”, она тоже отправляется в стэк, затем приходит переменная “а” так как операнды по приоритету польской формы стоят выше операций. И, наконец, приходит +. Далее, когда приходит закрывающая скобка, данные выгружаются из стэка в другой, буферный стэк, и приходит в пустой стэк операнд с затем операция “\*”. После они тоже выгружаются в буферный стэк и начинается вычисления с помощью функции Calculate, которая просит нас ввести значения переменных и вычисляет по польской форме.

# Заключение

В итоге мы имеет совокупность классов, которые позволяют вычислять некоторые арифметические выражения, контролировать корректность ввода и позволяют проводить вычисления над одним выражением, но с разными переменными. Решение такой простой задачи сильно упрощает жизнь людям в самых разных областях деятельности: от вычисления школьных выражений, до сложных инженерных расчётов. Каждый класс покрыт набором юнит тестов в изоляции. Кроме этого, написаны интеграционные тесты, проверяющие корректность работы несколько классов вместе. Получены навыки работы со стеком, синтаксического разбора выражения, анализа на корректность и вычисления выражения.

# Литература

1. <http://habrahabr.ru/post/100869/>
2. Лекции В.П.Гергель

# Приложения

Код калькулятора и преобразователя в польскую форму.

|  |
| --- |
| #include "Calculate.h"  using namespace std;  string postfix\_note(string exp){  Stack<char> opStack;  Stack<char> trackStack;  map <char, int> op;  op['\*'] = 3;  op['/'] = 3;  op['+'] = 2;  op['-'] = 2;  op['('] = 1;  op['='] = 0;  char buff;  for (int i = 0; i < exp.length(); i++){  buff = exp[i];  if (op.count(buff)) {  if ((!opStack.IsEmpty()) && (op[buff] <= op[opStack.Peek()]) && (buff != '('))  while ((!opStack.IsEmpty()) && (op[buff] <= op[opStack.Peek()]))  trackStack.Push(opStack.Put());    opStack.Push(buff);  continue;  }  if (((buff >= 0x41) && (buff <= 0x5A))  || ((buff >= 0x61) && (buff <= 0x7A))) {  trackStack.Push(buff);  continue;  }  if (buff == ')') {  while ((!opStack.IsEmpty()) && (opStack.Peek() != '('))  trackStack.Push(opStack.Put());  if (opStack.IsEmpty())  throw "не правильно расставлены скобки";  opStack.Put();  continue;  }  throw "недопустимый символ";  }  while (!opStack.IsEmpty()){  if (opStack.Peek() == '(')  throw "не правильно расставлены скобки";  trackStack.Push(opStack.Put());  }  if (trackStack.IsEmpty())  throw "нет данных";  string result;  string tmp;  while (!trackStack.IsEmpty()) {  tmp = trackStack.Put();  result.insert(0, tmp);  }  return result;  }  double Calculate(string exp)  {  if (exp == "")  throw "нет данных";  Stack<float> trackStack;  char buff;  double leftOperand;  double rightOperand;  map<char, double> val;  for (int i = 0; i < exp.length(); i++){  buff = exp[i];  if (exp[exp.length() - 1] == '=')  val[exp[0]] = 0;  if (((buff >= 0x41) && (buff <= 0x5A))  || ((buff >= 0x61) && (buff <= 0x7A))){  if (!val.count(buff)){  cout << '\t' << buff << " = ";  cin >> val[buff];  }  trackStack.Push(val[buff]);  continue;  }  if (trackStack.IsEmpty())  throw "нет совпадения с числом операндов";  rightOperand = trackStack.Put();  if (trackStack.IsEmpty())  throw "нет совпадения с числом операндов";  leftOperand = trackStack.Put();  switch (buff){  case '+':{trackStack.Push(leftOperand + rightOperand); break; }  case '-':{trackStack.Push(leftOperand - rightOperand); break; }  case '\*':{trackStack.Push(leftOperand \* rightOperand); break; }  case '/':{trackStack.Push(leftOperand / rightOperand); break; }  }  }  float result = trackStack.Put();  if (!trackStack.IsEmpty())  throw "Стэк переполнен";  return result;  } |