Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Вычисление арифметических выражений.

Выполнил:

студент ИИТММ гр. 381703-1

Девликамов В.О

Проверил:

Ассистент кафедры МОСТ ИИТММ

Волокитин В.Д.

Нижний Новгород

2018 г.

Содержание

Введение……………………………………………………………………………………3

Постановка задачи…………………………………………………………………………4

Руководство пользователя………………………………………………………………...5

Руководство программиста………………………………………………………………..6

Описание структуры программы………………………………………………………6

Описание структур данных……………………………………………………………..7

Описание алгоритмов…………………………………………………………………...7

[Заключение…………………………………………………………………………………8](#_Toc499930791)

Литература……………………………………………………………………………….....9

Приложения……………………………………………………………………………….10

# Введение

# Дана строка, представляющая собой математическое выражение, содержащее числа, переменные, различные операции. Требуется вычислить его значение за O(n) , где n — длина строки.

Обратная польская нотация — это форма записи математических выражений, в которой операторы расположены после своих операндов.

Обратная польская нотация была разработана австралийским философом и специалистом в области теории вычислительных машин Чарльзом Хэмблином в середине 1950-х на основе польской нотации, которая была предложена в 1920 г. польским математиком Яном Лукасевичем.

Описанный алгоритм реализован в данной лабораторной работе.

# Постановка задачи

Разработать программу, выполняющую вычисление арифметического выражения с вещественными числами. Выражение в качестве операндов может содержать переменные и вещественные числа. Допустимые операции известны: +, -, /, \*. Допускается наличие знака "-" в начале выражения или после открывающей скобки. Опционально - наличие математических функций (sin, соs, ln, exp, и т.д.) Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю вид ошибки и номера символов строки, в которых были найдены ошибки.

# Руководство пользователя

Необходимо ввести корректное арифметическое выражение (правильно расставлены скобки, вещественные числа должны разделяться точкой, параметры могут быть только маленький латинские символы, допустимые операции (+, -, \*, /)). Пользователь может ввести параметр (любая буква латинского алфавита), также может ввести функцию (sin, cos, tan, arctan, log, sqrt). На выходе пользователь получает посчитанное выражение, или сообщение об ошибке (вид ошибки и номер символа строки)

# Руководство программиста

**Описание структур программы.**

Программа состоит из 4-x проектов: gtest, sample, tests, ariphmetiс.

gtest содержит реализацию Google Test.

sample содержит реализацию пользовательского консольная приложения.

tests – содержит (test\_main.cpp – запускает все тесты, test\_stack.cpp – тесты для стека, test\_ariphmetic.cpp – тесты для проверки всех функций из ariphmetic.cpp)

ariphmetic – содержит реализацию алгоритма.

**Описание структур данных**

**main\_arithmetic.cpp**

string s – строка, которую вводит пользователь

**arithmetic.cpp**

pair <string, int> error – пара строка – содержит вид ошибки, число – содержит номер символа строки

Stack <pair<char, int>> temp – стек для проверки правильной скобочной последовательности

char v – значение i символа строки

Stack <Lexem> s, s1 – стеки для перевода в обратную польскую нотацию

map <string, int> perem – map для запоминания значений переменных

set <string> was – set для проверки был ли ранее введен такой же параметр

**arithmetic.h**

enum Lex\_Type – для определения типа введенного символа

bool may\_unary – определяет унарная ли операция (true – унарная, false –бинарная)

int pos – позиция символа в строке

**Описание алгоритмов**

Перевод строки в массив лексем:

Избавляемся от всех пробелов, для каждого элемента строки определяем вид лексемы и помещаем его в массив лексем

Перевод в обратную польскую нотацию происходит следующим способом:

1. Если лексема это число, то кладем во 2 стек
2. Если лексема это операция, то перекладываем из 1 стека во второй, пока приоритет текущей лексемы меньше или равна приоритету лексемы на вершине стека
3. Если лексема это открывающая скобка – то кладем в стек 1
4. Если лексема это закрывающая скобка – то перекладываем из стека 1 во 2 стек, пока не встретим открывающуюся скобку

Вычисление по обратной польской нотации происходит следующим способом:

1. Если лексема это число, то кладем в стек
2. Если лексема это бинарная операция – достаем из стека 2 числа, вычисляем и кладем обратно в стек
3. Если лексема это унарная операция – достаем из стека число, вычисляем и кладем обратно в стек

В конце в стеке останется одно число, которое и будет результатом посчитанного выражения.

# Заключение

В результате проделанной работы были достигнуты следующие результаты:

1. Разработан шаблонный класс Stack
2. Написан алгоритм, вычисляющий значение арифметического выражения
3. Функции тестируются с помощью Google Tests.

# Литература

1. Разбор выражения.Обратная польскаю нотация [<https://e-maxx.ru/algo/expressions_parsing>]
2. Обратная польская запись [https://ru.wikipedia.org/wiki/Обратная\_польская\_запись]

**Приложения**

1. double Calculate::calc()
2. {
3. std::map<std::string, int> perem;
4. std::set<std::string> was;
5. Prover1();
6. Prover2();
7. Prover3();
8. PerevodVPol();
9. Stack <double> calculate(size);
10. for (int i = 0; i < size; i++)
11. {
12. Lex\_Type v = lex[i].tp;
13. if (v == VALUE)
14. {
15. std::stringstream temp;
16. temp << lex[i].s;
17. double ans;
18. temp >> ans;
19. calculate.push(ans);
20. }
21. else if (v == Perem)
22. {
23. if (was.find(lex[i].s) == was.end())
24. {
25. std::cout << "Enter the variable value " << lex[i].s << " : ";
26. double x;
27. std::cin >> x;
28. calculate.push(x);
29. was.insert(lex[i].s);
30. perem[lex[i].s] = x;
31. }
32. else
33. {
34. calculate.push(perem[lex[i].s]);
35. }
36. }
37. else
38. {
39. if (lex[i].may\_unary)
40. {
41. if (v == FUNC)
42. {
43. double a = calculate.front();
44. calculate.pop();
45. calculate.push(val\_func(lex[i].s, a, lex[i].pos));
46. }
47. else
48. {
49. double a = calculate.front();
50. calculate.pop();
51. calculate.push((-1)\*a);
52. }
53. }
54. else {
55. double a = calculate.front();
56. calculate.pop();
57. double b = calculate.front();
58. calculate.pop();
59. char temp = lex[i].s[0];
60. calculate.push(oper(b, a, temp, lex[i].pos));
61. }
62. }
63. }
64. return calculate.front();
65. }