Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий математики и механики

Специальность прикладная математика и информатика

Отчёт по лабораторной работе

Вычисление арифметических выражений

Выполнил:

Студент ПМИ гр. 381703-1

Арефьев А. М.

Проверил:

Ассистент кафедры МОСТ ИИТММ

Волокитин В. Д.

Нижний Новгород

2018 г.

Оглавление

[Введение 3](#_Toc499325868)

[Постановка задачи 4](#_Toc499325869)

[Руководство пользователя 5](#_Toc499325870)

[Руководство программиста 6](#_Toc499325871)

[Описание структур данных 6](#_Toc499325872)

[Описание алгоритмов 6](#_Toc499325873)

[Описание структуры программы 7](#_Toc499325874)

[Заключение 9](#_Toc499325875)

[Литература 10](#_Toc499325876)

[Приложения 11](#_Toc499325877)

# Введение

Данная лабораторная работа направлена на изучение интерпретатора математических выражений, а также на изучение алгоритмов, которые помогут мне в будущем.

Для выполнения этой работы нужно знать обратную польскую запись и разбор строки.

Обратная польская запись - форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Такая запись позволяет преобразовать математическое выражения со скобками в математическое выражение без скобок, и такое выражение очень удобно вычислить при помощи стеков.

Разбор строки нужен для разбора математического выражения на лексемы с которыми непосредственно работает программа.

Работа рассчитана на получение знаний об алгоритмах и о языке С++ в ходе ее написания. Данная работа помогает совершенствовать навык программирования на С++, навык работы с классами и работы с алгоритмами.

# Постановка задачи

Предлагается 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

1. Запустите программу.
2. Введите математическое выражение.

## Правила ввода математического выражения

# Выражение не должно содержать пробелов

# Переменные записываются как х1, х2 и так далее до х5

# Допустимы выражения sin, cos, ln, exp

# Их аргументы записываются в скобках

# Целая и дробная часть числа разделяется точкой

# Если целая часть числа равна 0, то перед точкой должен стоять 0

# Пример правильно написанного выражения:

# -ln(2.71)+12+(-12)+cos((9-9)\*5)-9/(3\*3)+0.25\*4

# Руководство программиста

## Описание структур данных

В программе используются класс лексем (lexem) и шаблонный класс стек (stack).

Класс лексем используется для создания массива лексем и стека лексем. Класс стек используется дляперевода в ОПЗ и вычисления выражения.

Лексема представляет из себя число или операцию. У класса лексем есть публичные методы которые позволяют узнать число это или операция, узнать приоритет операции, вернуть значение числа или операцию.

## Описание алгоритмов

1 Перевод в ОПЗ.

1. Вход: массив лексем, размер массиваN
2. В цикле от 1 до N (начиная с начала массива)
   1. Взять лексему
      1. Если это число то положить в новый массив лексем
      2. Если это операция то положить в стек, если он пустой,
         1. если стек не пустой, сравнить с приоритетами операций лежащих в стеке,
         2. если приоритет операции в стеке больше чем приоритет поступающей операции то достать операцию из стек , положить ее в новый массив.
3. После цикла добавить операции из стека в новый массив
4. Выход: массив лексем в ОПЗ

2.Проверка правильности математического выражения(proverka).

Вход: массив лексем, размер массиваN

В цикле от 1 до N брать лексему из массива проверять что следует после текущей лексемы:

|  |  |
| --- | --- |
| Текущая лексема | Следующая лексема |
| Число | Бинарная операция, ) |
| Бинарная операция | Унарная операция |
| Унарная операция (функция) | ( |
| ( | Число, унарная операция, ( |
| ) | Бинарная операция, ) |

В данной таблице показано что допускается после текущей лексемы, если что то не совпало, то выведется сообщение об ошибке.

Выход: сообщение о пройденной проверке

3. Вычисление(calculate).

1. Вход:массив лексем в ОПЗ
2. В цикле от 1 до N (начиная с начала массива)
   1. Достать лексему из массива
   2. Если лексема операнд, то добавить ее в стек
   3. Если бинарная операция, то достать из стека два значения, провести операцию, результат положить в стек
   4. Если унарная операция, то достать из стека значения, провести операцию, положить в стек
3. После цикла результат выражения будет лежать на вершине стека
4. Выход: результат математического выражения

4.Solve.

Вход: строка

Разобрать строку на лексемы, записывая лексемы в массив

Провести проверку правильности математического выражения (2 алгоритм)

Перевод в ОПЗ(1 алгоритм)

Вычислить(3 алгоритм)

Выход: результат математического выражения.

## Описание структуры программы

Библиотеки, использующиеся в программе:

1. iostream(Для организации ввода-вывода в языке программирования)
2. cmath (Для выполнения простых математических операций. );
3. stack.h(Класс стек)
4. arithmetic.h(Класс лексем, функция solve)
5. string(Класс строка)

# Заключение

В итоге удалось реализовать программу ,которая вычисляет значения математических выражений.

Реализовано следующее:

1. Шаблонный класс стек
2. Класс лексем
3. Алгоритм разбиения строки на лексемы
4. Алгоритм проверки
5. Алгоритм перевода в ОПЗ
6. Алгоритм вычисления математического выражения

# Литература

1. Т. Пратт, М. Зелковиц. Языки программирования: разработка и реализация— 4-е издание. — Питер, 2002. — 688 с

А. В. Ахо, Р. Сети, Д. Д. Ульман. Компиляторы: принципы, технологии и инструменты. М.: «Вильямс», 2003. С. 51

# Приложения

Перевод в ОПЗ

lexem \*arr2;

arr2 = newlexem[j];

stack<lexem> t;

int l=0 ;

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

if(arr[k].isnum()) {

arr2[l] = arr[k];

l++;

} else {

if(t.isempty()) {

t.push(arr[k]);

} else {

if(arr[k].getprior() == 1) {

t.push(arr[k]);

} else {

while(!(t.isempty()) && (arr[k].getprior() <= t.show().getprior())) {

if(t.show().getprior() == 1) {

t.pop();

break;

} else {

arr2[l]=t.pop();

l++;

}

}

if (arr[k].getprior() != 0) {

t.push(arr[k]);

}

}

}

}

}

Проверка

intproverka (lexem\* arr, intn) {

int j=0;

for(inti = 0; i<n; i++) {

if(arr[i].isnum()){

if(i!=n-1){

if(( arr[i+1].getop() == "+")&&(arr[i+1].getop() == "-")&&(arr[i+1].getop() == "\*")&&(arr[i+1].getop() == "/")&&(arr[i+1].getop() == ")")) {return 1;}

}

} else {

if(arr[i].getop()=="(") {

j++;

if(i==n-1) {return 7;}

if ((arr[i+1].isop()) && ((arr[i+1].getop() !="(")&&(arr[i+1].getop()!="sin")&&(arr[i+1].getop()!="cos")&&(arr[i+1].getop()!="ln")&&(arr[i+1].getop()!="exp")&&(arr[i+1].getop() !="un-"))) {return 2;}

}

elseif(arr[i].getop()==")") {

j--;

if(i!=n-1){

if ((arr[i+1].isnum()) || ((arr[i+1].getop() != "+")&&(arr[i+1].getop() != "-")&&(arr[i+1].getop() != "\*")&&(arr[i+1].getop() != "/")&&(arr[i+1].getop() != ")"))) {return 3;}

}

}

elseif ((arr[i].getop()=="sin")||arr[i].getop()=="cos"||arr[i].getop()=="ln"||arr[i].getop()=="exp") {

if(i==n-1){return 7;}

elseif(arr[i+1].isnum() || (arr[i+1].getop() !="(") ) {return 8;}

}

else {

if(i==n-1) {return 7;}

elseif ((arr[i+1].isop()) && ((arr[i+1].getop() !="(") && (arr[i+1].getop() !="un-") && (arr[i+1].getop() !="cos")&& (arr[i+1].getop() !="sin")&& (arr[i+1].getop() !="ln")&& (arr[i+1].getop() !="exp"))) {return 4;}

}

if (j<0){return 5;}

}

}

if(j!=0){return 6;}

return 0;}

Вычисление

double calculate (lexem\* arr ,intl) {

stack<lexem> s;

lexem t;

double x1, x2;

for(inti = 0; i<l; i++) {

if (arr[i].isnum()) {

s.push(arr[i]);

}

else {

if(arr[i].getop() == "+"){

x2 = (s.pop()).getnum();

x1 = (s.pop()).getnum();

t = x1 + x2;}

if(arr[i].getop() == "-"){

x2 = (s.pop()).getnum();

x1 = (s.pop()).getnum();

t = x1 - x2;}

if(arr[i].getop() == "\*"){

x2 = (s.pop()).getnum();

x1 = (s.pop()).getnum();

t = x1 \* x2;}

if(arr[i].getop() == "/"){

x2 = (s.pop()).getnum();

x1 = (s.pop()).getnum();

t = x1 / x2;}

if(arr[i].getop() == "un-"){

x2 = (s.pop()).getnum();

t = -1\*x2;}

if(arr[i].getop() == "sin"){

x2 = (s.pop()).getnum();

t = sin(x2);}

if(arr[i].getop() == "cos"){

x2 = (s.pop()).getnum();

t = cos(x2);}

if(arr[i].getop() == "ln"){

x2 = (s.pop()).getnum();

t = log(x2);}

if(arr[i].getop() == "exp"){

x2 = (s.pop()).getnum();

t = exp(x2);}

s.push(t);

}

}