

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

(ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки «Прикладная математика и информатика»

ОТЧЕТ

по лабораторной работе

Вычисление арифметических выражений

Выполнил(а): студент(ка) группы 381703-1

Кузнецова А.И.

_____Подпись

Проверил: ассистент кафедры МОСТ ИИТММ

Волокитин В. Д.

_____Подпись

Нижний Новгород

2018

Оглавление

Введение.....	3
1. Постановка учебно-практической задачи.....	4
2. Руководство пользователя.....	5
3. Руководство программиста.....	6
3.1 Описание алгоритмов вычисления арифметического выражения.....	6
3.2. Описание структур данных.....	9
3.3. Описание структура программного комплекса.....	11
Заключение.....	12
Список литературы.....	13
Приложение.....	14

Введение

Арифметическое выражение – это запись математической формулы с использованием констант, переменных, функций, знаков арифметических операций и круглых скобок.

Арифметические выражения являются важным элементом языков программирования высокого уровня, а их трансляция входит в функции языковых процессоров. Поскольку эти выражения являются неотъемлемой частью фактически всех вычислительных программ, в составе языковых процессоров необходимо иметь алгоритмы, распознающие арифметические выражения в тексте программы и вычисляющие их как можно быстрее и эффективнее.

В данной лабораторной работе мы рассмотрим наиболее часто используемый алгоритм перевода выражения как польская запись.

В отчете приводится постановка задачи вычисления арифметических выражений с использованием стека, описание алгоритмов вычислений, а также дается описание программы и правил ее использования, прилагается текст программы и результаты выполнения подсчетов.

1. Постановка учебно-практической задачи

Цель работы:

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, cos, 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.

Исходные данные:

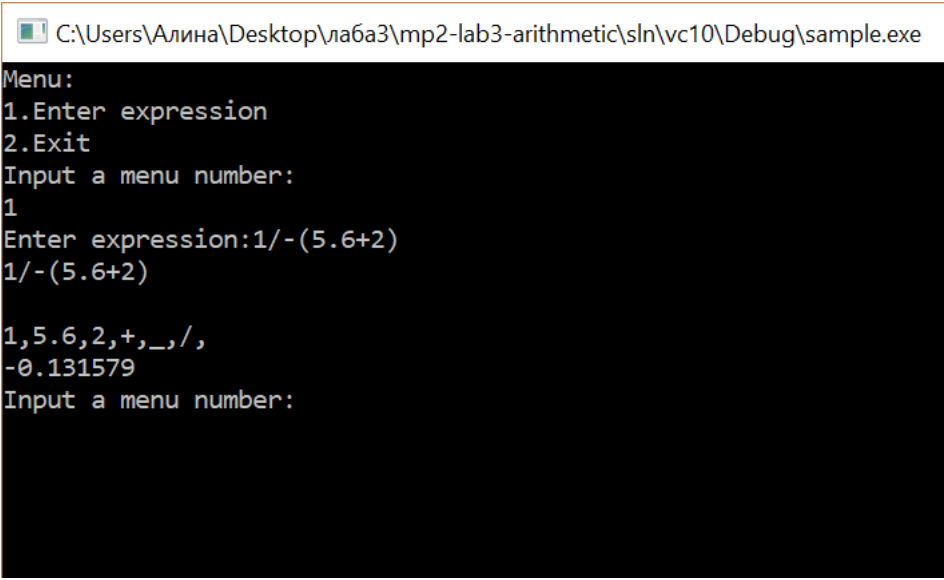
string s - строка, вводимая пользователем, представляющая собой арифметическое выражение.

Требуемый результат:

double res - число, результат вычисления выражения.

Контрольный пример:

Результат вычисления выражения $1/-(5.6+2)$ приведен на Рис.1.



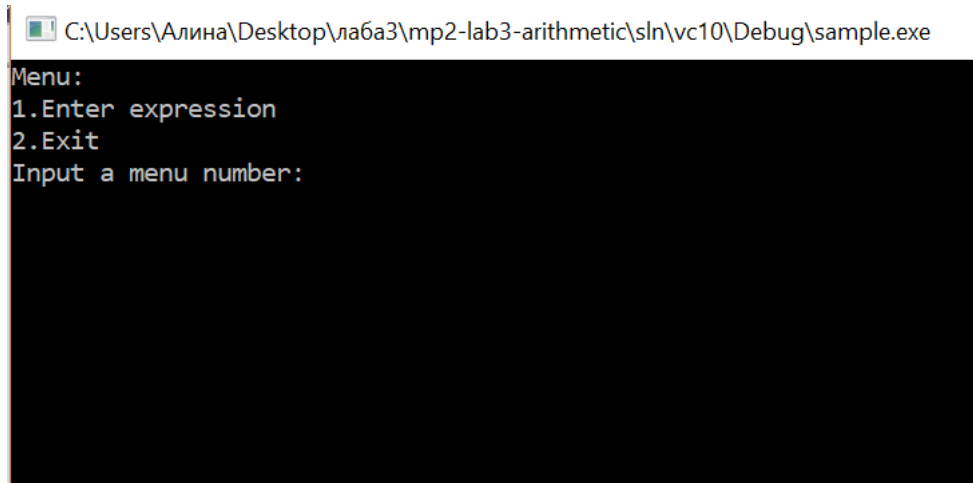
```
C:\Users\Алина\Desktop\лаба3\mp2-lab3-arithmetic\sln\vc10\Debug\sample.exe
Menu:
1.Enter expression
2.Exit
Input a menu number:
1
Enter expression:1/-(5.6+2)
1/-(5.6+2)

1,5.6,2,+,_,/,
-0.131579
Input a menu number:
```

Рис.1. Результат вычисления выражения

2. Руководство пользователя

Для вычисления арифметического выражения необходимо запустить файл sample.exe. На экране после запуска открывается главное меню(Рис.2).



```
C:\Users\Алина\Desktop\лаба3\mp2-lab3-arithmetic\sln\vc10\Debug\sample.exe
Menu:
1.Enter expression
2.Exit
Input a menu number:
```

Рис.2. Главное меню программы

После выбора 1-го пункта меню необходимо ввести арифметическое выражение, которое вам нужно посчитать. В качестве результата программа выводит полученный результат или сообщение об ошибке.

Для выхода из программы необходимо выбрать 2 пункт меню.

3. Руководство программиста

3.1 Описание алгоритмов вычисления арифметического выражения

В программах комплекса реализованы алгоритмы определения символа как оператора или операнда, установления приоритета операции, отличия унарного минуса от бинарного, перевода инфиксной строки в новую строку(с учетом унарного минуса), проверки корректности арифметического выражения, его перевода в постфиксную запись, вычисление и тестирование функций с помощью Google test-ов.

Приведем общее описание методов их выполнения.

Проверка корректности арифметического выражения

`bool Error()` - функция проверки корректности выражения, которая проверяет инфиксную строку, а после выводит сообщение об ошибке в выражении с комментарием, указывающим на тип ошибки и на выходе возвращает значение `false`(если ошибка имеется) или `true`(если нет ошибок).

Функция содержит в себе несколько циклов и условий.

Первое условие проверяет строку на пустоту. Если она пуста, то выводится сообщение:

"string is empty". Иначе проверяем начало и конец строки на наличие символов, которые не могут стоять в начале и конце строки и выводит соответствующие сообщения: "wrong begin of expression" или "wrong end of expression".

Первый цикл проверяет строку на ошибки ввода вещественных чисел(с точкой) и если допущена ошибка, то выводит сообщение: "spelling error".

Второй цикл проверяет, что после ' (' не может следовать число или точка. Если допущена такая ошибка выводит сообщение: "spelling error".

Третий цикл проверяет строку на следование операторов подряд(при этом допускается следование унарного минуса после операции). Если допущена такая ошибка выводит сообщение: "several operations in a row".

Четвертый цикл проверяет, что после ') ' не может следовать ' (' или точка. Если допущена такая ошибка выводит сообщение: "spelling error".

Пятый цикл проверяет строку на некорректные символы и если таковой имеется, то выводит сообщение об ошибке и позицию некорректного символа: "invalid symbol on i position". В этом же цикле проводится проверка строки на скобки. Создается стек, в который при проходе по строке помещается каждая открывающаяся скобка, которая извлекается из него как только встречается закрывающаяся скобка. Если в какой-то момент стек оказывается пустым, то выводится сообщение: "error in brackets".

Определение символа как оператора или операнда

`bool IsOperator(char s)` — функция принимает на вход символ, если он равен одному из символов: +, -, *, /, _ то этот символ - оператор и возвращается значение `true`, иначе возвращается `false`.

Установления приоритета операции

`int Priority(char s)` — функция принимает на вход символ. Самый большой приоритет имеет унарный минус(' _ '). Если символ - унарный минус, то возвращается значение 2. Если ' * ' или ' / ', то возвращается 1. Если ' + ' или ' - '(бинарный минус), то возвращается 0.

Отличие унарного минуса от бинарного

`char IsUnar(int i)` — функция принимает на вход позицию элемента. Если элемент первый в строке или предыдущий ' (' или оператор, то возвращаем ' _ ' иначе возвращаем ' - '.

Перевод инфиксной строки в новую строку(с учетом унарного минуса)

`string NewInfix(string s)` — функция принимает на вход строку и возвращает строку. В функции создается строка. Далее в цикле последовательно рассматривается каждый элемент строки. Если это не оператор, то записываем в новую строку. Если это унарный минус(' _ ') , то записываем в строку ' _ ', а если другой оператор — записываем его в строку.

Перевод арифметического выражения в обратную польскую запись

`string ToPostfix()` — функция работает с инфиксной строкой и возвращает постфиксную строку.

В функции создается стек для хранения скобок и операторов. Далее в цикле последовательно проверяем каждый элемент.

Константы кладутся в формируемую строку в порядке их появления в исходном массиве(при этом пропускаем пробелы в вводимой пользователем строке и выделяем вещественные числа). Элементы в формируемой записи разделяются запятыми.

При появлении операции в исходном массиве:

- a. если в стеке нет операций или верхним элементом стека является открывающая скобка, операции кладётся в стек;
- b. если новая операции имеет больший или равный приоритет, чем верхняя операции в стеке, то новая операции кладётся в стек;
- c. если новая операция имеет меньший приоритет, чем верхняя операции в стеке, то операции, находящиеся в стеке, до ближайшей открывающей скобки или до операции с приоритетом меньшим, чем у новой операции, перекладываются в формируемую запись, а новая операции кладётся в стек.

Открывающая скобка кладётся в стек.

Закрывающая скобка выталкивает из стека в формируемую запись все операции до ближайшей открывающей скобки, открывающая скобка удаляется из стека.

После того, как мы добрались до конца исходного выражения, операции, оставшиеся в стеке, перекладываются в формируемое выражение.

Вычисление арифметического выражения

`double Calculator()` — функция работает с постфиксной строкой, а возвращает результат вычисления выражения.

Создаем цикл для хранения чисел. По постфиксной строке идет цикл. Значения констант кладутся в стек(вещественные числа преобразуются с помощью функции `atof` из `string` в

double). Когда встречается операция, из стека берутся два верхних значения, вычисляется результат применения операции к этим значениям, и результат помещается в стек. Если встречается унарный минус, то берётся одно значение из стека, а результат помещается в стек на его место.

В конце выполнения цикла в стеке остается один элемент, значение которого равно результату вычисления арифметического выражения.

3.2. Описание структур данных

В ходе выполнения лабораторной работы был создан класс стек (class Stack). Интерфейс класса приведен ниже:

```
class Stack
{
    TType *Mass; //указатель на массив

    int Top; //вершина стека

    int Size; //размер стека

public:
    Stack(); //конструктор по умолчанию, создает стек размером в 10 элементов
    Stack(int Size); //конструктор с параметром, создает стек размера Size
    void Push(TType val); //помещение элемента на вершину стека
    TType Pop(); //извлечение элемента с вершины стека
    TType CheckTop(); //получение значения вершины стека
    TType GetSize(); //получение размера стека
    void ClearStack(); //очистка стека
    bool IsEmpty(); //проверка стека на пустоту
    bool IsFull(); //проверка стека на полноту
    void NewLen(); //перевыделение памяти при вставке элемента в полный стек
    TType CheckTopEl(); //просмотр верхнего элемента (без удаления)
    ~Stack(); //деструктор
};
```

В данной лабораторной работе также использовался класс арифметик (class Arithmetic). Интерфейс класса приведен ниже:

```
class Arithmetic
{
    string infix; //инфиксная строка

    string postfix; //постфиксная строка

public:
    Arithmetic(string s); //конструктор

    bool IsOperator(char s); //проверяет, является символ оператором или операндом

    int Priority(char s); //устанавливает приоритет операций
```

```
char IsUnar(int i); //отличает унарный минус от бинарного
string NewInfix(string s); //перезапись инфиксной строки с учетом унарного минуса
string ToPostfix(); //перевод в постфиксную запись
double count(double a, double b, char c); //вычисление выражения
double Unar(double a); //возвращает -a
double Calculator(); //вычисление арифметического выражения
bool Error(); // проверка на ошибки инфиксной строки
};
```

3.3. Описание структура программного комплекса

Программный комплекс представлен программой, состоящей из пяти файлов:

1. В файле `arithmetic.h` реализованы методы класса `Arithmetic` и функции.
2. В файле `stack.h` представлен интерфейс класса `Stack` и реализованы его методы.
3. В файле `main_arithmetic.cpp` находится основная программа.
4. Файл `test_stack.cpp` содержит тесты для класса `Stack`.
5. Файл `test_arithmetic.cpp` содержит тесты для класса `Arithmetic` и функций.
6. Файл `test_main.cpp` запускает все google tests.

Заключение

В лабораторной работе был разработан программный комплекс, включающий в себя собственный класс Stack и реализованы алгоритмы перевода строки в постфиксную запись, вычисления значения арифметического выражения, записанного в постфиксной форме; реализовали обработку ошибок синтаксиса инфиксной строки. Программный комплекс позволяет в режиме диалога вводить арифметическое выражение. В работе использовалась структура данных стек, что показало ее удобство. Методы классов и функции тестировались с помощью Google test-ов. Приведенные эксперименты доказали работоспособность данного программного комплекса.

Список литературы

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ. М.:МЦНМО, 1999.- 960 с., 263 ил.
2. <http://natalia.appmat.ru/c&c++/postfisso.html>

Приложение

```
class Arithmetic
{
    string infix;

    string postfix;

public:
    Arithmetic(string s)
    {
        infix = s;
    }

    bool IsOperator(char s)
    {
        if (s == '+' || s == '*' || s == '-' || s == '/' || s == '_')
            return true;

        else return false;
    }

    int Priority(char s)
    {
        if (s == '_')
            return 2;
        if (s == '*')
            return 1;
        if (s == '/')
            return 1;
        if (s == '+')
            return 0;
        if (s == '-')
            return 0;
    }

    char IsUnar(int i)
    {
        int j = i;
        if ((j==0) || infix[j - 1] == '(' || IsOperator(infix[j - 1]))
            return '_';
        else return '-';
    }

    string NewInfix(string s)
    {
        string t;
        for (int i = 0; i < s.size(); i++)
```

```

    {
        if (!IsOperator(s[i]))
            t += s[i];
        else
            if (IsUnar(i) == '_')
                t += '_';
            else t += s[i];
    }
    return t;
}

```

string ToPostfix()

```

{
    Stack<char> ops(infix.size());

    string tmp = NewInfix(infix);
    infix = tmp;

    int j = 0;

    for (int i = 0; i < infix.size(); i++)
    {
        if (!IsOperator(infix[i]) && infix[i] != '(' && infix[i] != ')')
        {
            if (infix[i] != ' ')
            {
                int j = i;

                while (!IsOperator(infix[j]) && j != infix.size() &&
infix[j] != ')') && infix[j] != ' ')
                {
                    postfix += infix[j];
                    j++;
                }
                i = j - 1;
                postfix += ','; //разделяю элементы запятыми
            }
        }

        else

            if (ops.IsEmpty() || ops.CheckTopEl() == '(')
                ops.Push(infix[i]);
            else

```

```

        if (Priority(infix[i]) >= Priority(opers.CheckTopEl()))
            opers.Push(infix[i]);
        else
            if (Priority(infix[i]) <
Priority(opers.CheckTopEl()))

                {
                    while ((!opers.IsEmpty()) &&
(opers.CheckTopEl() != '(' || (Priority(opers.CheckTopEl()) < Priority(infix[i]))))
                        {
                            postfix += opers.Pop();
                            postfix += ',';
                        }
                    opers.Push(infix[i]);
                }

            else
                if (infix[i] == '(')
                    opers.Push(infix[i]);
                else
                    if (infix[i] == ')')
                        {
                            while (opers.CheckTopEl() !=
'(')
                                {
                                    postfix += opers.Pop();
                                    postfix += ',';
                                }
                            opers.Pop();
                        }
            }
        while (!opers.IsEmpty())
        {
            char t = opers.Pop();
            if (t != '(' && t != ')')
            {
                postfix += t;
            }
        }
    }
}

```



```

        postfix += ',';
    }

}

return postfix;

}

double count(double a, double b, char c)
{
    double res;
    if (c == '+')
        res = a + b;
    if (c == '-')
        res = a - b;
    if (c == '*')
        res = a * b;
    if (c == '/')
    {
        if (b == 0) throw "impossible to divide by zero";
        res = a / b;
    }

    return res;
}

double Unar(double a)
{
    return (-a);
}

```

```

double Calculator()
{
    double res,k;
    Stack<double> nums(postfix.size());

    for (int i = 0; i < postfix.size(); i++)
    {
        if (postfix[i] == ',')
            goto metka;
        if (!IsOperator(postfix[i]))
        {
            string tmp;

            tmp += postfix[i];

            i++;

            while (postfix[i] != ',')
            {

                tmp += postfix[i];

                i++;
            }

```

```

    }

    nums.Push(atof(tmp.c_str())); // atof из string в double
}

else
    if (postfix[i] == '_')
    {
        double d = Unar(nums.Pop());
        nums.Push(d);
        i++;
    }
    else
    {
        k = nums.Pop();
        res = count(nums.Pop(), k, postfix[i]);
        nums.Push(res);
        i++;
    }
}

metka: return (nums.Pop());

}

bool Error() {
    bool res = true;

    int l = 0;

    Stack<char> x(infix.size());

    int t = 0;

    char brasc;

    if (infix.empty())
    {
        cout << " string is empty" << endl;

        return false;
    }

    else
    {
        if ((infix[0] == '+') || (infix[0] == '*') || (infix[0] == '/') ||
(infix[0] == ')') || (infix[0] == '.'))
        {
            res = false;

            cout << "wrong begin of expression" << endl;
        }
    }
}

```

```

        if (IsOperator(infix[infix.size() - 1]) || (infix[infix.size() - 1] ==
'(') || (infix[infix.size() - 1] == '.'))
        {
            res = false;
            cout << "wrong end of expression" << endl;
        }
    }

    for (int i = 0; i < infix.size() - 2; i++)
    {

        if (infix[i] == '.')
        {
            if (isdigit(infix[i + 1]) == 0)
            {
                cout << "spelling error" << endl;
                res = false;
            }
        }
    }

    for (int i = 0; i < infix.size() - 2; i++)
    {

        if (infix[i] == ')')
        {
            if ((isdigit(infix[i + 1]) != 0) || (infix[i + 1] == '.'))
            {
                cout << "spelling error" << endl;
                res = false;
            }
        }
    }

    for (int i = 0; i < infix.size() - 2; i++)

```

```

{

    if (infix[i] == '+' || infix[i] == '*' || infix[i] == '/')
    {
        if (infix[i + 1] == '+' || infix[i + 1] == '*' || infix[i + 1]
== '/')

        {
            cout << "several operations in a row" << endl;
            res = false;
        }
    }
}

for (int i = 0; i < infix.size() - 2; i++)
{

    if (infix[i] == '(')
    {
        if ((infix[i + 1] == ')') || (infix[i + 1] == '.'))
        {
            cout << "spelling error" << endl;
            res = false;
        }
    }
}

for (int i = 0; i < infix.size(); i++)
{
    if (!((isdigit(infix[i]) != 0) || (infix[i] == '(') || (infix[i] ==
'(') || IsOperator(infix[i]) || (infix[i] == '.') || (infix[i] == ' ')))
    {
        cout << "invalid symbol on " << i + 1 << " position" << endl;
        res = false;
    }
}

```

```

        if (infix[i] == '(')
            x.Push('(');
        else
            if (infix[i] == ')')
            {
                if (!(x.IsEmpty()))
                {
                    brasc = x.Pop();
                }
                else
                {
                    t++;
                }
            }
    }

    if (t != 0) {
        res = false;
        cout << "error in brackets" << endl;
    }

    return res;
}

};

```