Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

Национальный исследовательский Нижегородский государственный

университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчёт по лабораторной работе**

**«Вычисление арифметических выражений»**

Выполнил:

Студент группы 3821Б1ПМ2

Куршев А.А.

Проверил:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2022

**Содержание**

Постановка задачи………………………………………………………………………………….3

Руководство пользователя…………………………………………………………………………4

Руководство разработчика…………………………………………………………………………5

Методы реализации…………………………………………………………………………………7

Результаты проверки………………………………………………………………………………..8

Заключение…………………………………………………………………………………………..9

**Постановка задачи**

Цель данной лабораторной работы заключается в реализации шаблонного класса TStack и использования его для реализации метода перевода выражения из инфиксной формы в так называемую постфиксную(она же обратная польская нотация). Помимо этого необходимо реализовать тесты для проверки корректности работы программы.

**Руководство пользователя**

Сразу после запуска, программа выводит пользователю список поддерживаемых команд. Данная реализации поддерживает операции сложения, вычитания, умножения и деления. Также программа поддерживает указание приоритета операции посредством использования скобок.Далее программа ожидает ввода выражения в инфиксной форме. На этом этапе осуществляется проверка корректности выражения. Далее программа выведет на экран преобразованное выражение в постфиксной форме. Если выражение имеет не только числа, но и переменные, то программа попросит пользователя ввести значения имеющихся переменных. В итоге программа выводит на экран подсчитанное значение выражения.

**Руководство разработчика**

Данная программа имеет следующую структуру:

TStack - шаблонный класс стека, поддерживающий хранение 1 типа данных.

При создании экземпляра класса, который требует ввести начальный размер стека, создаётся массив pMem соответствующего размера, указатель на вершину стека ставится в значение -1, означающее, что стек пуст.

IsEmpty - метод, возвращающий истину, если стек пустой, в ином случае ложь.

IsFull - метод, проверяющий полноту стека, возвращает true, если стек полон, false - стек не полон.

clean - метод очистки стека.

Size - метод, возвращающий текущий размер стека.

get\_top - метод, возвращающий вершину стека, без удаления элемента из стека.

pop - метод взятия элемента с вершины стека, убирает верхний элемент.

push - принимает и “кладёт” элемент на вершину стека.

~TStack - деструктор, который очищает pMem.

Lexem - базовый абстрактный класс лексем. От него в дальнейшем наследуются все доступные лексемы: Операции, Числа и Переменные.

Вместе с классами наследниками имеет 4 поля:

type - хранит тип лексемы,

data - хранит строковое представление лексемы,

priority - хранит приоритет данной лексемы,

num - хранит числовое значение.

Методы:

const string GetData() const, const string GetType() const, const int GetPriority() const, const double GetNum() const - гетеры, возвращающие значения соответствующих полей.

SetNum - изменяет числовое значение лексемы, доступно только лексеме-наследнику переменной.

SetPriority - чистая, в базовом классе, функция, перегруженная в классах-наследниках, которая ставит приоритет данной лексеме в зависимости от её типа.

TPostfix - непосредственно класс постфиксного выражения.

Поля:

infix и postfix - соответственно инфиксное и постфиксное представление выражения,

infix\_lexems и postfix\_lexems - соответственно вектора, содержащие указатели на лексемы,

op\_count - число операций,

lex\_count - число лексем.

Методы:

IsOperation, IsNum, IsVar - методы, проверяющие является ли строка чем то из типов лексем.

parse - парсер выражения, разбивает строку на лексемы.

InfixToPostfix - переводит инфиксное представление в постфиксное.

check - проверяет корректность введённого выражения.

Calculate - вычисляет итоговое значение выражения.

GetResult - гетер, возвращающий значение итогового выражения.

~TPostfix - деструктор, удаляющий указатели на лексемы из векторов.

**Методы реализации**

Для реализации перевода в постфиксную форму был использован алгоритм Дейкстра.

Пока есть лексемы для чтения:

Читаем очередную лексему.

Если лексемой является числом или переменной, добавляем её к выходной строке.

Если лексема является открывающей скобкой, помещаем её в стек.

Если лексема является закрывающей скобкой:

До тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется. Если стек закончился раньше, чем мы встретили открывающую скобку, это означает, что в выражении либо неверно поставлен разделитель, либо не согласованы скобки.

Если лексема является бинарной операцией , тогда:

1) Если операция на вершине стека приоритетнее или такого же уровня приоритета

выталкиваем верхний элемент стека в выходную строку;

2) помещаем операцию в стек.

Когда входная строка закончилась, выталкиваем все лексемы из стека в выходную строку. В стеке должны были остаться только лексемы операций; если это не так, значит в выражении не согласованы скобки.

Также, алгоритм для вычисления итогового выражения:

Идём слева направо по вектору, который хранит постфиксное представление выражения:

Если лексема число или переменная приписываем её значение к строковому представлению постфиксной формы.

Если лексема операция, то необходимо взять с вершины стека значение, в количестве, которое необходимо для данной операции. После кинуть в стек результат этой операции.

**Результаты проверки**

Корректность работы данной программы была проведена с помощью тестов, реализованных на базе библиотеки Google Test.

**Заключение**

В ходе лабораторной работы был успешно реализован алгоритм перевода инфиксной формы в постфиксную. Также были успешно реализованы TStack, TPostfix и Lexm с классами-наследниками.