Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе №3**

**«Вычисление арифметических выражений»**

**Выполнил**:

студент группы 382006-1

Расчётнов А.С.

**Проверил**:

доцент кафедры МОСТ, к.т.н.,

Сысоев А.В.

Нижний Новгород

2021

Содержание

[Введение 3](#_Toc529541653)

[1. Постановка задачи 4](#_Toc529541654)

[2. Руководство пользователя 5](#_Toc529541655)

[3.1. Описание структуры программы 6](#_Toc529541656)

[3.2. Описание алгоритмов 10](#_Toc529541657)

[4. Результаты экспериментов 11](#_Toc529541658)

[Заключение 12](#_Toc529541659)

[Литература 13](#_Toc529541660)

[Приложение 14](#_Toc529541661)

# Введение

Лабораторная работа направлена на изучение стека – динамической структуры данных. Изучить стек предлагается при выполнении задачи о переводе арифметического выражения из инфиксной формы в постфиксную и последующем его вычислении.

# Постановка задачи

Разработать структуру данных стек и с помощью неё реализовать расчёт арифметических выражений с использованием обратной польской записи (постфиксной формы).

Выполнение работы предполагает решение следующих задач:

1. Разработка интерфейса шаблонного класса TStack.
2. Реализация методов шаблонного класса TStack.
3. Разработка интерфейса класса TPostfix для работы с постфиксной формой.
4. Реализация методов класса TPostfix.
5. Разработка и реализация тестов для классов TStack и TPostfix на базе Google Test.
6. Публикация исходных кодов в личном репозитории на GitHub.

# Руководство пользователя

При открытии программы пользователь видит окно с приглашением к вводу арифметического выражения (рис.1).

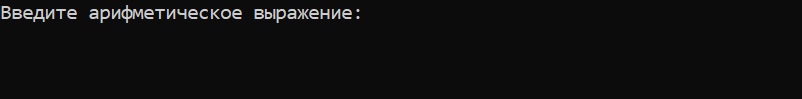


Рис.1. Стартовое окно

Арифметическое выражение необходимо вводить без пробелов, используя только арифметические операции '+', '-', '\*', '/', буквы латинского алфавита, а также математические скобки '(', ')'. Операнды могут быть как односимвольными, так и многосимвольными.

После ввода арифметического выражения, на экран выводится информация, которая содержит само арифметическое выражение, его постфиксную форму, а также приглашение к инициализации переменных (рис.2).

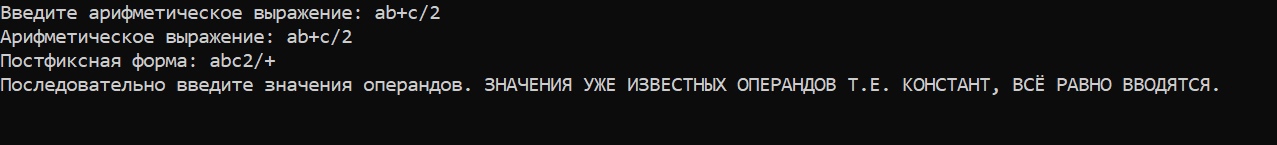


Рис.2 Окно с информацией после ввода арифметического выражения

Стоит обратить внимание на предложение, написанное заглавными буквами: значения уже известных операндов всё равно вводятся. Пример: арифметическое выражение: a + 2. Для того, чтобы посчитать его результат, необходимо ввести как значение переменной a (произвольное), так и значение операнда 2 (значение двойки). После ввода значений, на экран выведется результат (рис.3).

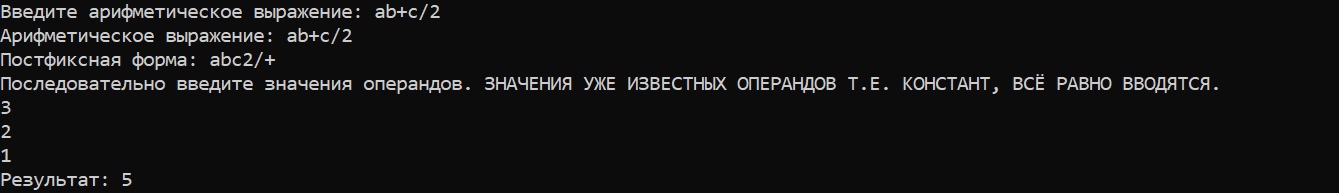


Рис.3. Результат арифметического выражения

1. **Руководство программиста**

## Описание структуры программы

Программа реализована на языке программирования C++ и состоит из 4 файлов: postfix.cpp, postfix.h, sample\_postfix.cpp, stack.h. Также проект содержит программу с модульными тестами для проверки корректности работы методов классов, программа тестирования содержит 3 файла: test\_main.cpp, test\_tpostfix.cpp, test\_tstack.cpp.

Файл postfix.h содержит прототип класса TPostfix, в файле postfix.cpp реализованы методы класса TPostfix. stack.h содержит реализацию шаблонного класса стека TStack. В файле sample\_postfix.cpp расположен код для проверки работоспособности класса TPostfix.

Поля класса TStack:

* «T\* pMem»

Данное поле хранит содержимое стека произвольного типа T. Поле по умолчанию при создании объекта класса инициализируется -1.

* «int size»

Поле хранит размер стека (размер выделенной памяти).

* «int top»

Поле хранит индекс вершины стека.

Методы класса TStack:

* «TStack(int \_size)»

Метод является конструктором класса, поле size инициализируется значением параметра \_size, затем выделяется память размера size.

* «TStack(int \_size, T value)»

Ещё один конструктор, выполняет всю ту же самою работу, что и предыдущий, вдобавок сразу же добавляя в стек объект типа value.

* «TStack(const TStack& object)»

Конструктор копирования.

* «~TStack()»

Деструктор класса, освобождает память, содержащуюся в T\* pMem.

* «void push(const T& object)»

Метод добавляет объект в стек.

Входные данные: ссылка на объект типа T – object.

Выходных данных не имеет.

* «int stack\_size()»

Метод возвращает значение поля size.

Входные данных не имеет.

Выходные данные: переменная типа int, содержащая значение поля size.

* «int current\_size()»

Метод возвращает текущее количество объектов в стеке.

Входных данных не имеет.

Выходные данные: переменная типа int – количество объектов в стеке.

* «bool is\_empty()»

Метод проверяет стек на пустоту.

Входных данных не имеет.

Выходные данные: переменная типа bool, содержащая нуль, если стек пуст и единицу в противном случае.

* «bool is\_full()»

Метод проверяет стек на полноту.

Входных данных не имеет.

Выходные данные: переменная типа bool, содержащая единицу, если стек полон и нуль в противном случае.

* «T back()»

Метод возвращает значение вершины стека.

Входных данных не имеет.

Выходные данные: переменная типа T, содержащая значение вершины стека.

* «T pop()»

Метод возвращает значение вершины стека и удаляет её оттуда.

Входных данных не имеет.

Выходные данные: переменная типа T, содержащая значение вершины стека.

* «bool operator==(const TStack& object) const»

Метод является перегруженным оператором == (сравнения) для данного класса.

Входные данные: константная ссылка объект типа TStack – object – объект для сравнения.

Выходные данные: переменная типа bool, содержащая единицу, если объекты равны и нуль в противном случае.

* «bool operator!=(const TStack& object) const»

Метод является перегруженным оператором != (сравнения) для данного класса.

Входные данные: константная ссылка объект типа TStack – object – объект для сравнения.

Выходные данные: переменная типа bool, содержащая единицу, если объекты не равны и нуль в противном случае.

* «TStack& operator=(const TStack& object)»

Метод является перегруженным оператором = (присваивания) для данного класса.

Входные данные: константная ссылка объект типа TStack – object.

Выходные данные: ссылка на тип TStack.

Поля класса TPostfix:

* «string infix»

Поле хранит инфиксную форму арифметического выражения в виде строки.

* «vector<string> postfix»

Поле хранит постфиксную форму арифметического выражения в виде вектора строк.

Методы класса TPostfix:

* «vector<string> infix\_divide()»

Метод переводит инфиксную форму арифметического выражения из формата строки в формат вектора строк, разделяя её на символы. Каждая ячейка вектора хранит один символ. Метод находится в секции private.

Входных данных не имеет.

Выходные данные: вектор типа string, содержащий посимвольно разделённую инфиксную форму арифметического выражения.

* «vector<string> to\_postfix()»

Метод переводит арифметическое выражение из инфиксной формы в постфиксную. Метод находится в секции private.

Входных данных не имеет.

Выходные данные: вектор типа string, содержащий посимвольно разделённую постфиксную форму арифметического выражения.

* «string postfix\_to\_string()»

Метод переводит постфиксную форму из формата вектора строк в формат строки. Метод находится в секции private.

Входных данных не имеет.

Выходные данные: переменная типа string, хранящая постфиксную форму арифметического выражения.

* «bool correct\_input\_check()»

Метод проверяет корректность переданной при создании объекта в конструктор арифметического выражения (строки). Метод находится в секции private.

Входных данных не имеет.

Выходные данные: переменная типа bool, содержащая единицу, если арифметическое выражение корректно и нуль в противном случае.

* «TPostfix(string \_infix = "") : infix(\_infix)»

Метод является конструктором класса. Поле infix по умолчанию инициализируется значением "" или значением параметра \_infix, если он передан. Также, если проверка на корректность параметра \_infix проходит успешно (проверка осуществляется с помощью метода correct\_input\_check), то postfix инициализируется возвращаемым значением метода to\_postfix.

* «void set\_infix(string \_infix)»

Метод устанавливает поле infix.

Входные данные: переменная типа string - \_infix.

Выходных данных не имеет.

* «string get\_infix() const»

Метод возвращает значение поля infix.

Входных данных не имеет.

Выходные данные: переменная типа string – значение поля infix.

* «string get\_postfix()»

Метод возвращает значение поля postfix.

Входных данных не имеет.

Выходные данные: переменная типа string – значение поля postfix.

* «int operands\_counter()»

Метод возвращает количество операндов в арифметическом выражении.

Входных данных не имеет.

Выходные данные: переменная типа int – количество операндов в арифметическом выражении.

* «double calculate(vector<double> values)»

Метод возвращает посчитанное значение арифметического выражения.

Входные данные: вектор значений double для инициализации операндов арифметического выражения.

Выходные данные: переменная типа double – посчитанное значение арифметического выражения.

## Описание алгоритмов

* «infix\_divide»

Метод разделяет поле infix (строку) на операнды и операции, а также и сохраняет её как вектор строк. Разделение происходит путём посимвольной проверки строки. Если символ строки – операнд или операция, то он попадает в вектор строк. Если же символ – пробел, он пропускается и не попадает в вектор строк. По окончании алгоритма, метод возвращает вектор строк, содержащий разделённую на операнды и операции инфиксную форму арифметического выражения без пробелов.

* «to\_postfix»

Метод переводит арифметическое выражение из инфиксной формы в постфиксную. Перевод осуществляется путём прохода по вектору строк инфиксной формы и проверке каждого элемента вектора. Если элемент вектора – операнд, то он помещается в вектор постфиксной формы. Если элемент – операция, то он добавляется в стек и вынимается из стека в случае, если следующая добавленная в стек операция имеет приоритетность не превосходящую данной. Если элемент вектора – закрывающаяся скобка, то из стека операций вынимаются и добавляются в вектор постфиксной формы все операции, содержащиеся в стеке, вплоть до открывающейся скобки, сама открывающаяся скобка из стека вынимается, но в вектор постфиксной формы не добавляется. По окончании работы алгоритма, метод возвращает постфиксную форму арифметического выражения в виде вектора строк.

* «correct\_input\_check»

Метод осуществляет проверку корректности переданной в конструктор при создании объекта строки. Проверка осуществляется путём поэлементного прохода по вектору строк, который содержит разделённую на операции и операнды инфиксную форму арифметического выражения. Если элемент является открывающей или закрывающей скобками, то увеличивается счётчик. В конце цикла, счётчик обязан быть кратен двум, что обеспечивает одинаковое количество открывающих и закрывающих скобок. Если проверка успешна, метод возвращает единицу, в противном случае метод возвращает нуль.

* «calculate»

Метод считает значение арифметического выражения, с введёнными в него значениями операндов. Происходит поэлементный проход по постфиксной форме выражения, если элемент – операнд, он помещается в стек. Если элемент – операция – из стека операндов вынимаются два операнда и операция применяется к ним, результат помещается обратно в стек. По окончании алгоритма в стеке останется единственное значение – значение арифметического выражения.

# Результаты экспериментов

Введённое арифметическое выражение: a+b-c, введённые значения переменных: 1, 2, 3 (рис.4).

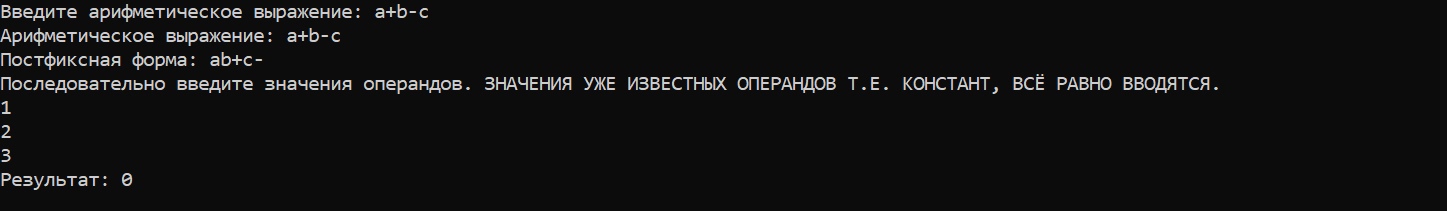


Рис.4. Результат работы программы

# Заключение

В результате лабораторной работы был реализован класс TStack и класс TPostfix, с помощью которых осуществлялся перевод арифметического выражения в постфиксную форму и его вычисление.

# Литература

# Бьёрн Страуструп. «Язык программирования C++»:

<http://www.8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>.

1. Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В.. Лабораторный практикум (по программе "Алгоритмы и структуры данных "):

<http://www.lib.unn.ru/students/020302.html>.

# Приложение

**Postfix.h**

class TPostfix  
{  
 std::string infix;  
 std::vector<std::string> postfix;  
private:  
 std::vector<std::string> infix\_divide();  
 std::vector<std::string> to\_postfix();  
 std::string postfix\_to\_string();  
 bool correct\_input\_check();  
public:  
 TPostfix(std::string \_infix = "") : infix(\_infix)  
 {  
 if (!correct\_input\_check())  
 throw "incorrect input.";  
 postfix = to\_postfix();  
 }  
 void set\_infix(std::string \_infix);  
 std::string get\_infix() { return infix; }  
 std::string get\_postfix();  
 int operands\_counter();  
 double calculate(std::vector<double> values); // Ввод переменных, вычисление по постфиксной форме  
};

**Postfix.cpp**

bool TPostfix::correct\_input\_check() {  
 int counter = 0;  
 std::vector<std::string> tmp(infix\_divide());  
 for (int i = 0; i < tmp.size(); i++)  
 if (tmp[i] == "(" || tmp[i] == ")")  
 counter++;  
 if (!(counter % 2))  
 return true;  
 else  
 return false;  
}