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

**Отчет по лабораторной работе**

**«Вычисление арифметических выражений»**

**Выполнила**:

Студентка группы 3822Б1ПМ1

Жесткова А.Е.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2023

Содержание

[Введение 3](#_Toc529541653)

[1. Постановка задачи 4](#_Toc529541654)

[2. Руководство пользователя 5](#_Toc529541655)

[3.1. Описание структуры программы 7](#_Toc529541656)

[3.2. Описание алгоритмов 10](#_Toc529541657)

[4. Результаты экспериментов 11](#_Toc529541658)

[Заключение 12](#_Toc529541659)

[Литература 13](#_Toc529541660)

[Приложение 14](#_Toc529541661)

# Введение

Иногда пользователям необходимо быстро и правильно решить сложные арифметические уравнения. Обычный алгоритм реализации калькулятора слишком прост: пользователь выбирает операцию, вводит два числа, над которыми будет совершатся операция, и получает результат. В таком калькуляторе нельзя написать сложное арифметическое выражение (под сложным выражением понимают выражение с несколькими операциями).

Чтобы упростить вычисления, необходимо разработать программу, которая будет принимать на вход строку со сложным выражением и выводить результат. Для обработки такой строки существует алгоритм обратной польской записи (или же постфиксная запись), реализованный с помощью стека. Данный алгоритм переводит инфиксную запись строки в постфиксную и производит вычисления.

# Постановка задачи

Цель работы: реализация на языке С++ структуры данных Стек для использование в подсчёте арифметических выражений с использованием обратной польской записи (постфиксной формы).

Основные задачи лабораторной работы:

1. Разработка шаблонного класса TStack и реализация его методов;
2. Разработка класса TPostfix для работы с обратной польской записью и реализация его методов;
3. Разработка и реализация тестов для классов TStack и TPostfix на базе Google Test.

# Руководство пользователя

После запуска программы пользователю необходимо ознакомиться с правилами работы с программой, а затем ввести алгебраическое выражение, которое необходимо вычислить:

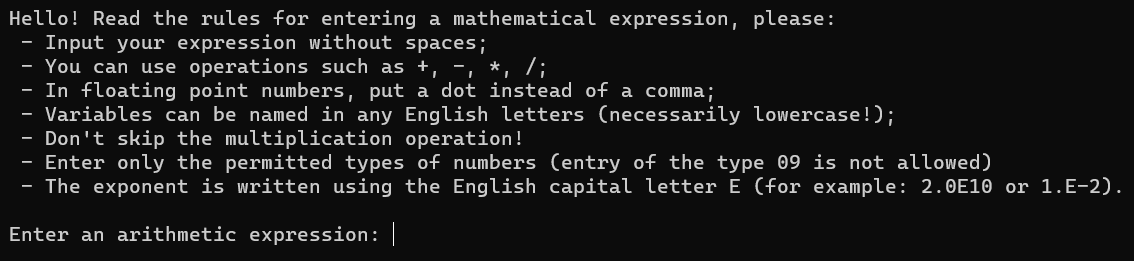


Рис. . «Главное меню: условия работы с программой»

Далее у пользователя спрашивают, хочет ли он посчитать этот пример. Он может выбрать «да», «нет» или «ввести новое выражение» (необходимо ввести цифру выбранного варианта):



Рис. . «Главное меню: запрос на подсчёт выражения»

Если пользователь выбрал вариант «Yes» (см. Рис. 2), то программа выдаст следующие результаты: инфиксная и постфиксная запись алгебраического выражения и результат:

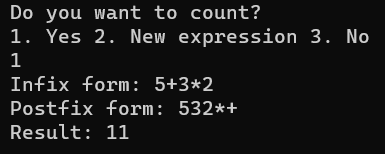


Рис. . «Результат работы программы»

Если пользователь ввёл выражение с переменными, программа перед подсчётом попросит ввести значения этих переменных:

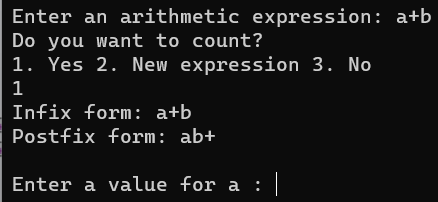


Рис. . «Работа с переменными»

После ввода значений переменных пользователь получит результат вычислений и вопрос: «хочет ли он дальше работать с этим выражением?»:

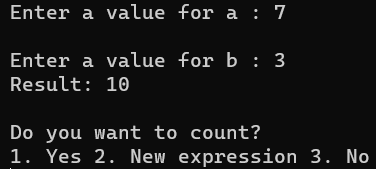


Рис. . «Результат работы с переменными»

Если пользователь выбрал вариант «New expression» (см. Рис. 2), то программа предложит ввести новое выражение:

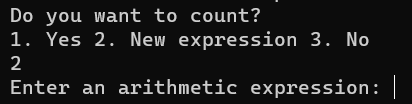


Рис. . «Повторный запрос на подсчёт выражения 1»

Если же пользователь выбрал вариант «No» (см. Рис. 2), то программа заканчивает свою работу:

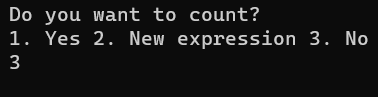


Рис. . «Повторный запрос на подсчёт выражения 2»

Бывают случаи, когда пользователь вводит некорректное арифметическое выражение. Тогда программа укажет, что за ошибка и на каком она месте:

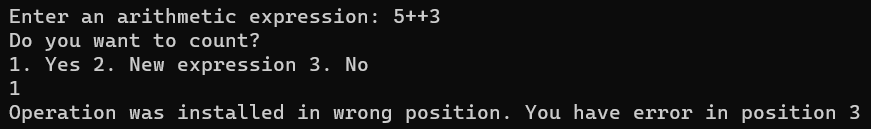


Рис. . «Ошибка при вводе алгебраического выражения»

1. **Руководство программиста**

## Описание структуры программы

Вся программа расположена в папке mp2-2023-lab3-postfix. В данной работе существует 4 файла с основным кодом программы, 3 файла с тестами, которые проверяют корректность работы кода, и дополнительные файлы, не требующие редакции:

1. ***stack.h (расположение: mp2-2023-lab3-postfix\include\stack.h)***

Реализация шаблонного класса TStack. Private область содержит поля класса:

* T\* pMem;
* int ind = -1;
* size\_t size;

В Public области содержатся методы класса:

* **Stack()**: конструктор по умолчанию. Создает объект класса, выделяя память на 5 элементов шаблонного типа.
* **Stack(size\_t n)**: конструктор с параметром. Создаёт объект класса, выделяя память на n элементов шаблонного типа.
* **void push(T elem)**: добавление элемента на вершину стека. Если для нового элемента недостаточно места, функция изменяет размер массива, чтобы разместить его. Это достигается путем создания нового массива с удвоенным исходным размером и копирования существующих элементов перед добавлением нового элемента в следующую доступную позицию. После завершения этих операций функция удваивает значение size для дальнейшего использования.
* **T pop()**: извлечение верхнего элемента из стека. Сначала идёт проверка на пустоту стека. Если он пуст, то бросается ошибка, иначе возвращаем верхний элемент и уменьшаем размер стека на 1.
* **T top()**: возврат верхнего элемента их стека. Сначала идёт проверка на пустоту стека. Если он пуст, то бросается ошибка, иначе возвращается верхний элемент стека.
* **bool isEmpty() const noexcept**: проверка на пустоту. Возвращает значение true, если поле ind равно -1, иначе - false.
* **size\_t GetSize()**: получение количества элементов в стеке. Возвращает значение ind + 1.
* **void clear()**: очистка стека. Присваивает переменной ind значение -1.
* **~Stack()**: деструктор класса. Очищает выделенную память у pMem.

1. ***arithmetic.h (расположение: mp2-2023-lab3-postfix\include\arithmetic.h)***

Реализация класса TPostfix. Private область содержит поля класса:

* size\_t size;
* string infix, postfix;
* string\* data;
* TStack<double> numbers;
* double result;
* string error;

В Public области содержатся методы класса, которые реализуются в файле arithmetic.cpp.

1. ***arithmetic.cpp (расположение: mp2-2023-lab3-postfix\src\arithmetic.cpp)***

Реализация методов класса TPostfix. Файл содержит следующие функции:

* **TPostfix(string new\_infix)**: конструктор с инициализацией. Инициализирует infix значением new\_infix, size – new\_infix.size() (функция возвращает размер строки). Остальные поля инициализируются по умолчанию. У data память выделяется по размеру size. В конце вызывается метод toLexem() для перевода строки infix в массив строк.
* **void toLexem()**: перевод строки infix в массив лексем, а также вывода ошибок о неправильной инфиксной записи. Цикл проходит по infix и с помощью условий if, else if и else определяет лексему и проверяет ее на корректность ввода. В случае ошибки, вызывается исключение и указывается позиция с ошибкой. Размер массива изменяется на количество лексем.
* **void value\_of\_variable()**: присваивание значения переменным из алгебраического выражения. Происходит проход по массиву лексем для поиска переменных. Если найдена, то пользователь вводит переменную в виде строки. Далее с текущего индекса массива лексем идет поиск этой же переменной. Все переменные заменяются на введенное пользователем значение.
* **size\_t getPriority(string operat) const**: получение приоритета для операции. На вход подается операция и при помощи условий if, else if и else каждой операции присваивается приоритет – какая по счёту следует данная операция.
* **void toPostfix()**: перевод инфиксной строки в постфиксную форму. Происходит проход по массиву лексем: если это операция, то помещаем её в стек, если число, то новый массив pform. Если приоритет операции меньше чем у верхнего элемента стека, то выталкиваем все элементы в массив pform. В конце копируем элементы pform в data и удаляем массив pform.
* **double string\_in\_double(string number)**: перевод строки в тип данных double. Происходит проход по string, умножая на 10 символ infix[i]-‘0’ каждую итерацию. Если находит ‘.’, то начинает считать количество символов после нее. Значение делится на 10 в степени k – количество символов после точки. Если в массиве найдем символ E (экспонента), то все последующие символы складываются в string e\_num. Программа вызывается рекурсивно. Значение передается в переменную e\_double. Значение double\_number умножается на 10 в степени e\_double.
* **void toCalculate()**: вычисление постфиксной формы. Происходит проход по массиву data: если элемент массива – число, то к нему применяется функция *string\_in\_double(string number)* и перемещение в стек numbers*.* Если же элемент – операция, то из стека вытаскивается число, перемещается в переменную temp, затем вытаскивается еще одно число из стека и выполняется соответствующая операция. Значение кладётся обратно в стек. Последнее значение из стека извлекается и кладётся в поле result.
* **string getPostfix() const**: получение поля postfix. Возвращаемое значение - string postfix.
* **string getInfix() const**: получение поля infix. Возвращаемое значение - string infix.
* **double getResult() const**: получение поля result. Возвращаемое значение – double result.
* **bool isDigit(const char& elem)**: проверка элемента на операцию. Если это операция – возвращается значение true, иначе – false.
* **~TPostfix()**: деструктор класса TPostfix. Удаляет выделенную память у data.

1. ***main\_arithmetic.cpp (расположение: mp2-2023-lab3-postfix\samples\main\_arithmetic.cpp)***

Пользовательское приложение:

* **int main()**: операции cout выводят правила пользования программой. Строка infix будет хранить введенное значение пользователем. В блоке try catch происходит обработка введенного выражения и вывод результатов, а также обработка исключений.

1. ***test\_main.cpp, test\_ arithmetic.cpp, test\_stack.cpp (расположение: mp2-2023-lab3-postfix\test)***

Тесты для проверки корректности работы кода.

## Описание алгоритмов

1. ***Стек***

Это структура данных, организованная по принципо LIFO: last in first out. Главные методы данной структуры: push и pop. Push кладет элемент на вершину стека, а pop - возвращает элемент с вершины стека.

1. ***Обратная польская запись (постфиксная форма)***

Алгоритм создания такой записи:

1. Пока в инфиксной записи есть символы для чтения:
   1. Читаем символ в строке.
   2. Если символ является числом, то добавляем его к выходной строке.
   3. Если символ является открывающей скобкой, то помещаем его в стек.
   4. Если символ является закрывающей скобкой, то до тех пор, пока верхним элементом стека не станет открывающая скобка, извлекаем элементы из стека и перемещаем в выходную строку (при этом открывающая скобка в выходную строку не добавляется, а просто удаляется из стека).
   5. Если символ является бинарной операцией Х, то:
      1. Пока на вершине стека префиксная функция ИЛИ операция на вершине стека приоритетнее или такого же уровня приоритета как у Х: выталкиваем верхний элемент стека в выходную строку.
      2. Помещаем операцию Х в стек.
2. Когда входная строка закончилась, выталкиваем все символы из стека в выходную строку. В стеке должны были остаться только символы операций; если это не так, значит в выражении не согласованы скобки.
3. ***Вычисление постфиксной формы***

Алгоритм вычисления выражения в постфиксной форме основан на работе со стеком:

1. Обработка входного символа:
   1. Если символ является операндом, то он кладётся на вершину стека.
   2. Если символ является операцией, то она выполняется над необходимым количеством элементов, извлечённых их стека. Результат вычисления кладётся на вершину стека.
2. Пока входной набор полностью не обработается, перейти к шагу 1.
3. После полной обработки входного набора символов результат вычисления лежит на вершине стека.

# Результаты экспериментов

Результатом лабораторной работы стало приложение для вычисления алгебраических выражений, которое принимает инфиксную форму выражения, а возвращает постфиксную форму и результат вычисления. Данное приложение даёт возможность ввода алгебраических выражений с использованием нескольких операций, переменных и экспоненциальной записи числа. Корректность работы данного приложения и вычислений подтверждают юнит тесты (см Рис. 9 и Рис. 10 в разделе «Приложения»).

# Заключение

Мне удалось разработать структуру данных стек и с помощью нее реализовать расчет арифметических выражений с использованием обратной польской записи (постфиксной формы). Был разработано удобное меню с прописанными правилами для работы с приложением. Также мною были написаны юнит тесты для подтверждения корректности работы кода.

# Литература

1. Сайт Хабр, стек - [О стеке простыми словами — для студентов и просто начинающих / Хабр (habr.com)](https://habr.com/ru/articles/341586/)
2. Сайт Хабр, обратная польская запись - [Обратная польская запись / Хабр (habr.com)](https://habr.com/ru/articles/100869/)

# Приложение

void TPostfix::toPostfix() //converting infix to postfix

{

size\_t psize = 0;

string\* pform = new string[size];

TStack<string> operations;

for (size\_t i = 0; i < size; i++)

{

if (data[i] < "0" || data[i] == "~")

{

if (data[i] == "(")

operations.push(data[i]);

else if (data[i] == ")")

{

while ((operations.top() != "("))

pform[psize++] = operations.pop();

operations.pop();

}

else

{

if (!(data[i] == "~"))

{

while ((!operations.isEmpty()) && (getPriority(operations.top()) >= getPriority(data[i])))

pform[psize++] = operations.pop();

}

operations.push(data[i]);

}

}

else pform[psize++] = data[i];

}

while (!operations.isEmpty())

pform[psize++] = operations.pop();

size = psize;

for (size\_t i = 0; i < size; i++)

{

data[i] = pform[i];

postfix += pform[i];

}

delete[] pform;

}

double TPostfix::string\_in\_double(string number) //сonverting a string to a double

{

double double\_number = 0.0, sign = 1.0, e\_double = 0.0;

int flag = 0, k = 0, dot = 0; //k - the number of digits after the dot

string e\_num;

for (size\_t i = 0; i < number.size(); i++)

{

if (dot > 1)

{

error += "There is can't be more than one dot in the number. You have error in position ";

error += i + 1 + '0';

throw invalid\_argument(error.c\_str());

}

if (number[i] == '~')

{

if (i != 0)

{

error += "Minus can't stay in this place. You have error in position ";

error += i + 1 + '0';

throw invalid\_argument(error.c\_str());

}

sign = -1.0;

continue;

}

if (number[i] == '.')

{

flag = 1;

dot++;

continue;

}

if (number[i] == 'E')

{

for (size\_t j = i + 1; j < number.size(); j++)

{

if ((number[j] >= '0' && number[j] <= '9') || number[j] == '~' || number[j] == '-')

e\_num += number[j];

else

{

error += "Exponential notation contains foreign characters. You have error in position ";

error += i + 1 + '0';

throw invalid\_argument(error.c\_str());

}

}

e\_double = string\_in\_double(e\_num);

break;

}

if ((number[i] < '0') || (number[i] > '9'))

{

error += "Inccorect symbol. You have error in position ";

error += i + 1 + '0';

throw invalid\_argument(error.c\_str());

}

if (flag) k++;

double\_number = double\_number \* 10.0 + static\_cast<double>(number[i] - '0');

}

for (int i = 0; i < k; i++)

double\_number /= 10.0;

if (sign == -1.0)

double\_number \*= sign;

double\_number \*= pow(10.0, e\_double);

return double\_number;

}

void TPostfix::toCalculate() //calculations

{

value\_of\_variable();

double tmp;

double exp = pow(10, -20);

for (size\_t i = 0; i < size; i++)

{

if (data[i] == "+")

{

tmp = numbers.pop();

tmp = numbers.pop() + tmp;

numbers.push(tmp);

}

else if (data[i] == "-")

{

tmp = numbers.pop();

tmp = numbers.pop() - tmp;

numbers.push(tmp);

}

else if (data[i] == "\*")

{

tmp = numbers.pop();

tmp = numbers.pop() \* tmp;

numbers.push(tmp);

}

else if (data[i] == "/")

{

if (abs(numbers.top()) < exp)

{

error += "Division by zero isn't possible. You have error in position ";

error += i + '0';

throw invalid\_argument(error.c\_str());

}

tmp = numbers.pop();

tmp = numbers.pop() / tmp;

numbers.push(tmp);

}

else if (data[i] == "~")

{

tmp = numbers.pop();

numbers.push(-tmp);

}

else numbers.push(string\_in\_double(data[i]));

}

result = numbers.pop();

}

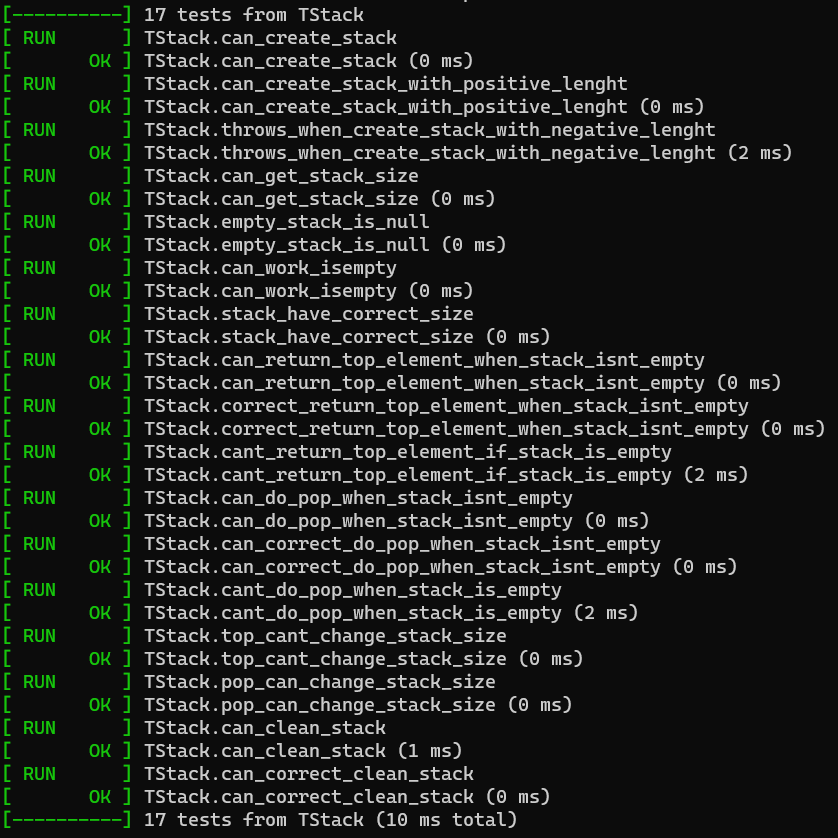


Рис. 9 «Результаты тестов по TStack»

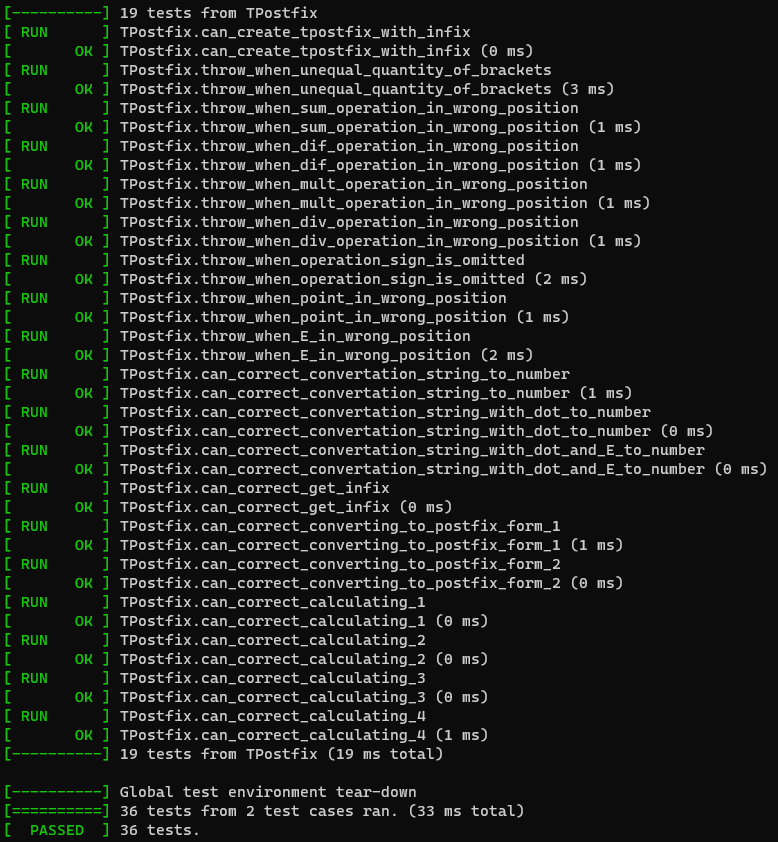


Рис. 10 «Результаты работы тестов по TPostfix»