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**

Выполнил:

студент гр. 381806 – 1

Пронькин Д.А.

Проверил:

к.т.н., доцент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2019 г.

Содержание

[Введение 3](#_Toc27999540)

[Постановка задачи 4](#_Toc27999541)

[Руководство пользователя 5](#_Toc27999542)

[Руководство программиста 7](#_Toc27999543)

[Описание структуры программы 7](#_Toc27999544)

[Описание структур данных 7](#_Toc27999545)

[Класс Stack 7](#_Toc27999546)

[Класс Converter 8](#_Toc27999547)

[Описание алгоритмов 9](#_Toc27999548)

[Алгоритмы структуры Stack 9](#_Toc27999549)

[Перевод в польскую нотацию 10](#_Toc27999550)

[Вычисление выражения записанного в польской нотации 12](#_Toc27999551)

[Заключение 14](#_Toc27999552)

[Литература 15](#_Toc27999553)

[Приложения 16](#_Toc27999554)

[Приложение 1. Основная функция 16](#_Toc27999555)

[Приложение 2. Класс Stack 16](#_Toc27999556)

[Приложение 3. Класс Converter 17](#_Toc27999557)

# Введение

Целью лабораторной работы является изучение и практическое применение структуры Стек. Темой данной работы является разработка программы, позволяющей вычислять значения произвольного арифметического выражения при помощи структуры Стек

# Постановка задачи

Задача: разработать программу, переводящую вводимое пользователем с клавиатуры арифметическое выражение, в польскую нотацию и вычисляющую его значение.

Входные данные: строка, содержащее арифметическое выражение, значение переменных содержащихся в выражение.

Выходные данные: строка, содержащая исходной выражение в польской нотации, значение выражения.

# Руководство пользователя

После запуска программы откроется консольный интерфейс с предложением ввести арифметическое выражение.

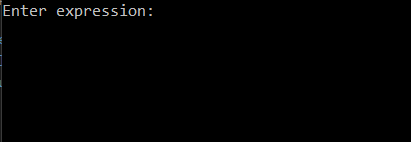


Рис. 1 Стартовый экран

После введения выражения нажмите клавишу Enter. Если ваше выражение корректно, то программа выведет польскую нотацию и предложить указать значение введенных переменных.

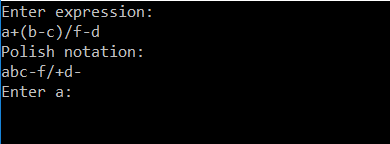


Рис. 2 Программы вывела польскую нотацию и ожидает ввода переменной a

Как только вы введете значения всех переменных, программа вычислит выражение и выведет результат

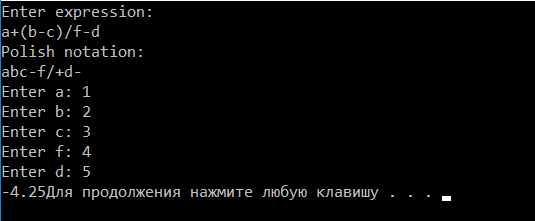


Рис. 3 Выведен результат

Если вы введете некорректное выражение или значение переменной при которой выражение не вычисляется, то программа выведет сообщение об ошибке и завершит работу.

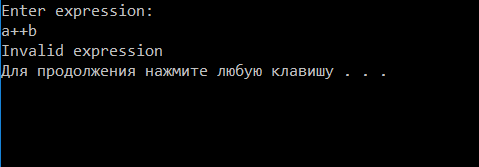


Рис. 4 Введено неверное выражение

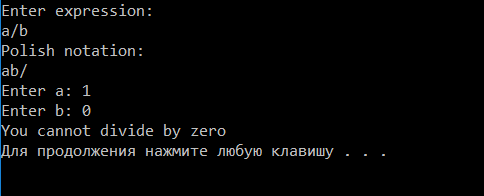


Рис. 5 Попытка деления на ноль

# Руководство программиста

## Описание структуры программы

Stack.h – класс, реализующий функционал структуры данных Стек.

Converter.h, Converter.cpp–модуль со статическим классом, содержащий методы для работы с арифметическими выражениями.

## Описание структур данных

### Класс Stack

Stack – класс, реализующий функционал структуры данных стек.

Объявление:

template<class ValueType>

classTStack {

public:

ValueType\* elems;

int size;

int top;

public:

TStack(int size = 4);

TStack(constTStack&);

~TStack();

void Push(ValueType);

ValueTypePop();

boolIsEmpty() const;

boolIsFull() const;

ValueTypeTopElems() const;

};

**Описание методов класса**

TStack(int size = 4)

Назначение: конструктор, инициализация объекта

Входные параметры: размер стека

Возвращаемое значение: нет

TStack(constTStack&)

Назначение: конструктор копирования, создание копии текущего стека

Входные параметры: константная ссылка на копируемый объект

Возвращаемое значение: нет

~TStack()

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

voidPush(ValueType)

Назначение: добавление значения в стек

Входные параметры: добавляемое значение

Возвращаемое значение: нет

ValueTypePop()

Назначение: извлечение элемента из стека

Входные параметры: нет

Возвращаемое значение: извлекаемое значение

boolIsEmpty() const

Назначение: проверка стека на пустоту

Входные параметры: нет

Возвращаемое значение: логическая переменная

boolIsFull() const

Назначение: проверка стека на полноту

Входные параметры: нет

Возвращаемое значение: логическая переменная

ValueTypeTopElems() const

Назначение: просмотр значения на вершине стека

Входные параметры: нет

Возвращаемое значение: значение на вершине стека

### Класс Converter

Converter – класс, реализующий методы для работы с арифметическими выражениями

**Объявление**

static class Converter {

private:

staticbool Priority(char, char);

static double Calculator(double, double, char);

public:

static string CreatePostFixForm(const string&);

static double Calculate(const string&, double\*, int);

static double\* GetValueOperands(const string&, int\*);

};

**Описание методов класса**

static string CreatePostFixForm(const string&)

Назначение: создание польской нотации

Входные параметры: исходное выражение

Возвращаемое значение: преобразованное в польскую нотацию исходное выражение

static double Calculate(const string&, double\*, int)

Назначение: вычисление выражения при помощи польской нотации

Входные параметры: исходное выражение, записанное в польской нотации, массив значений переменных, количество переменных

Возвращаемое значение: результат вычисления выражения

static double\* GetValueOperands(const string&, int\*)

Назначение:получение значений переменных

Входные параметры: исходное выражение, записанное в польской нотации, ссылка на переменную которая содержит количество переменных

Возвращаемое значение: массив значений переменных

staticbool Priority(char, char)

Назначение: сравнение приоритета пришедшей операции и операции на вершине стека

Входные параметры: 2 значения операций

Возвращаемое значение: логическая переменная

staticdoubleCalculator(double, double, char)

Назначение: калькулятор

Входные параметры: значения переменных и арифметическое действие в типе char

Возвращаемое значение: результат вычисления

## Описание алгоритмов

### Алгоритмы структуры Stack

Стек ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) stack — стопка; читается стэк) — [абстрактный тип данных](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), представляющий собой [список элементов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), организованных по принципу [LIFO](https://ru.wikipedia.org/wiki/LIFO) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) lastin — firstout, «последним пришёл — первым вышел»).

Характеристики стека:

1. Массив данных
2. Индекс последнего занятого
3. Размер стека

Методы стека:

1. **Добавление элемента в стек:**если стек не полный, то увеличить значение индекса последнего не занятого на единицу и положить по этому индексу элемент в массив данных
2. **Получение данных на вершине стека:** если стек не пустой, то возвращается значение по индексу последнего занятого
3. **Удаление данных:** если стек не пустой, то возвращается значение по индексу последнего занятого и индекс последнего занятого уменьшается на единицу
4. **Проверка на пустоту:** если значение последнего занятого равно -1, то стек пуст
5. **Проверка на полноту:** если значение последнего занятого равно размеру стека – 1, то стек полон

### Перевод в польскую нотацию

На вход приходит строка содержащая выражение, на выходе получается строка содержащая польскую нотацию

Перед выполнением алгоритма создаем два стека, в (1) стек будем записывать операнды, во (2) стек будем записывать операции

**Алгоритм:**

1. Если пришел операнд, то записываем в стек (1)
2. Если пришла левая скобка, то записываем в стек (2)
3. Если пришла правая скобка, то перезаписываем все значения из стека (2) в стек (1), пока не встретится левая скобка, которую тоже удаляем
4. Если пришла операция, то сравниваем ее приоритет с приоритетом операции на вершине стека
5. Если пришла операция с более высоким приоритетом чем на вершине стека, то записываем ее в стек (2)
6. Если приоритет пришедшей операции ниже или равен приоритету операции на вершине стека, то перезаписываем все операции с большим или равным приоритетом из стека (2) в стек (1), пока не встретим левую скобку или не дойдем до конца стека или не встретим операцию с меньшим приоритетом
7. После того как дошли до конца строки перекладываем все значения из стека (2) в стек (1), а затем при помощи конкатенации строк формируем польскую нотацию.

Рассмотрим в качестве примера выражение S\*(Y+N)/G

Таблица 1 Помещаем операнд в стек 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 |  |  |  |  |  |  |  |  |  |
| Стек 2 | S |  |  |  |  |  |  |  |  |

Таблица 2 Помещаем операцию в стек 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 | \* |  |  |  |  |  |  |  |  |
| Стек 2 | S |  |  |  |  |  |  |  |  |

Таблица 3 Помещаем открывающую скобку в стек 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 | \* | ( |  |  |  |  |  |  |  |
| Стек 2 | S |  |  |  |  |  |  |  |  |

Таблица 4 Помещаем операнд в стек 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 | \* | ( |  |  |  |  |  |  |  |
| Стек 2 | S | Y |  |  |  |  |  |  |  |

Таблица 5 Помещаем операцию в стек 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 | \* | ( | + |  |  |  |  |  |  |
| Стек 2 | S | Y |  |  |  |  |  |  |  |

Таблица 6 Помещаем операнд в стек 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 | \* | ( | + |  |  |  |  |  |  |
| Стек 2 | S | Y | N |  |  |  |  |  |  |

Таблица 7 Перемещаем все операции из стека 1 в стек 2 пока не встречаем открывающую скобку

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 | \* |  |  |  |  |  |  |  |  |
| Стек 2 | S | Y | N | + |  |  |  |  |  |

Таблица 8 Помещаем операцию в стек 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 | \* | / |  |  |  |  |  |  |  |
| Стек 2 | S | Y | N | + |  |  |  |  |  |

Таблица 9 Помещаем операнд в стек 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 | \* | / |  |  |  |  |  |  |  |
| Стек 2 | S | Y | N | + | G |  |  |  |  |

Таблица 10 Перемещаем все операции из стека 1 в стек 2(записано в обратном порядке)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выражение | S | \* | ( | Y | + | N | ) | / | G |
| Стек 1 |  |  |  |  |  |  |  |  |  |
| Стек 2 | S | Y | N | + | G | / | \* |  |  |

### Вычисление выражения записанного в польской нотации

На вход алгоритма подаётся выражение, записанное в польской нотации и значения переменных. На выходе имеем значение выражения.

Перед началом работы создаётся стек, в который будут подаваться промежуточные результаты, и на вершине которого в результате работе алгоритма окажется значение выражения.

В ходе алгоритма выражение просматривается слева направо, в зависимости от того, поступил оператор или операнд проводятся действия:

Если поступил операнд, то кладём в стек.

Если поступил оператор, то изымаем два значения из стека и проводим над ними операцию в порядке второй-первый.

В результате работы на вершине стека лежит значение выражения.

Рассмотрим в качестве примера SYN+G/\*

Таблица 11 Помещаем операнд в стек

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Об. п. н. | S | Y | N | + | G | / | \* |
| Стек | S |  |  |  |  |  |  |

Таблица 12 Помещаем операнд в стек

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Об. п. н. | S | Y | N | + | G | / | \* |
| Стек | S | Y |  |  |  |  |  |

Таблица 13 Помещаем операнд в стек

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Об. п. н. | S | Y | N | + | G | / | \* |
| Стек | S | Y | N |  |  |  |  |

Таблица 14 Складываем 2 верхних операнда и результат помещаем в стек

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Об. п. н. | S | Y | N | + | G | / | \* |
| Стек | S | Y+N |  |  |  |  |  |

Таблица 15 Помещаем операнд в стек

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Об. п. н. | S | Y | N | + | G | / | \* |
| Стек | S | Y+N | G |  |  |  |  |

Таблица 16 Выполняем деление

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Об. п. н. | S | Y | N | + | G | / | \* |
| Стек | S | (Y+N)/G |  |  |  |  |  |

Таблица 17 Выполняем умножение

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Об. п. н. | S | Y | N | + | G | / | \* |
| Стек | S\*(Y+N)/G |  |  |  |  |  |  |

# Заключение

В ходе выполнения лабораторной работы, была изучена структура данных стек, её принципы работы и возможный операции. Рассмотрен и реализован на практике алгоритм перевода выражения в постфиксную форму с последующим вычислением значения оного.

# Литература

Горьков Алексей Обратная польская запись:[<https://habr.com/ru/post/100869/>] 03.08.2010.

# Приложения

## Приложение 1. Основная функция

void main() {

Converter c;

string s, g;

int f = 1;

int\* k = &f;

cout << "Enter expression:" << endl;

cin >> s;

try {

string g = c.CreatePostFixForm(s);

cout << "Polish notation:" << endl;

cout << g << endl;

double \*V = new double[1];

char \*B = new char[1];

c.GetValueOperands(g, V, B, \*k);

cout << "Expression value: " << c.Calculate(g, V, B, \*k) << endl;

}

catch (char\* f) {

cout << f << endl;

}

}

## Приложение 2. Класс Stack

template<class ValueType>

class TStack {

public:

ValueType\* elems;

int size;

int top;

public:

TStack(int size = 4);

TStack(const TStack&);

~TStack();

void Push(ValueType);

ValueType Pop();

bool IsEmpty() const;

bool IsFull() const;

ValueType TopElems() const;

};

template<class ValueType>

TStack<ValueType>::TStack(int \_size) {

size = \_size;

top = -1;

elems = new ValueType[size];

}

template<class ValueType>

TStack<ValueType>::TStack(const TStack &\_copy) {

size = \_copy.size;

top = \_copy.top;

elems = new ValueType[size];

for (int i = 0; i < top + 1; i++) elems[i] = \_copy.elems[i];

}

template<class ValueType>

TStack<ValueType>::~TStack() {

size = 0;

top = -1;

delete[] elems;

}

template<class ValueType>

bool TStack<ValueType>::IsEmpty() const{

if (top == -1) return true;

return false;

}

template<class ValueType>

bool TStack<ValueType>::IsFull() const{

if (top == size - 1) return true;

return false;

}

template<class ValueType>

void TStack<ValueType>::Push(ValueType \_elem) {

if (this->IsFull()) throw "Error:Stack full";

elems[++top] = \_elem;

}

template<class ValueType>

ValueType TStack<ValueType>::Pop() {

if (this->IsEmpty()) throw "Error:Stack empty";

return elems[top--];

}

template<class ValueType>

ValueType TStack<ValueType>::TopElems() const {

return elems[top];

}

## Приложение 3. Класс Converter

static class Converter {

private:

static bool Priority(char, char);

static double Calculator(double, double, char);

public:

static string CreatePostFixForm(const string&);

static double Calculate(const string&, double\*, char\*, int);

static void GetValueOperands(const string&, double\*&, char\*&, int&);

};

string Converter::CreatePostFixForm(const string &\_str) {

int l = 0;

int r = 0;

for (int i = 0; i < \_str.length(); i++) {

if (\_str[i] == '(') l++;

if (\_str[i] == ')') r++;

}

if (l != r) throw "Invalid expression";

if ((\_str[0] == '+') || (\_str[0] == '-') || (\_str[0] == '\*') || (\_str[0] == '/')) throw "Invalid expression";

if (\_str[\_str.length() - 1] == '+' || \_str[\_str.length() - 1] == '-' || \_str[\_str.length() - 1] == '\*' || \_str[\_str.length() - 1] == '/') throw "Invalid expression";

for (int i = 0; i < \_str.length(); i++)

if (\_str[i] == '0' || \_str[i] == '1' || \_str[i] == '2' || \_str[i] == '3' || \_str[i] == '4' || \_str[i] == '5' || \_str[i] == '6' || \_str[i] == '7' || \_str[i] == '8' || \_str[i] == '9')

throw "Invalid expression";

for (int i = 1; i < \_str.length(); i++)

if ((\_str[i] != '+') && (\_str[i] != '-') && (\_str[i] != '\*') && (\_str[i] != '/') && (\_str[i] != '(') && (\_str[i] != ')') && (\_str[i - 1] != '+') && (\_str[i - 1] != '-') && (\_str[i - 1] != '\*') && (\_str[i - 1] != '/') && (\_str[i - 1] != '(') && (\_str[i - 1] != ')'))

throw "Invalid expression";

for (int i = 1; i < \_str.length(); i++)

if (((\_str[i] == '+') || (\_str[i] == '-') || (\_str[i] == '\*') || (\_str[i] == '/')) && ((\_str[i - 1] == '+') || (\_str[i - 1] == '-') || (\_str[i - 1] == '\*') || (\_str[i - 1] == '/')))

throw "Invalid expression";

TStack<double> operands(\_str.length());

TStack<double> operators(\_str.length());

string PostFixForm = "";

for (int i = 0; i < \_str.length(); i++) {

bool b = (\_str[i] == '+') || (\_str[i] == '-') || (\_str[i] == '\*') || (\_str[i] == '/') || (\_str[i] == '(') || (\_str[i] == ')');

if (!b) {

operands.Push(\_str[i]);

continue;

}

if (b) {

if (\_str[i] == ')') {

while (operators.TopElems() != '(')

operands.Push(operators.Pop());

operators.Pop();

continue;

}

if (Priority(\_str[i], operators.TopElems())) {

while ((Priority(\_str[i], operators.TopElems())) && !(operators.IsEmpty()) && (operators.TopElems() != '('))

operands.Push(operators.Pop());

operators.Push(\_str[i]);

continue;

}

operators.Push(\_str[i]);

}

}

while (!operators.IsEmpty())

operands.Push(operators.Pop());

while (!operands.IsEmpty())

operators.Push(operands.Pop());

while (!operators.IsEmpty())

PostFixForm += operators.Pop();

return PostFixForm;

}

double Converter::Calculate(const string &\_str, double\* VO, char\* B, int k) {

TStack<double> e(k);

int l = \_str.length();

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

if (\_str[i] == '+' || \_str[i] == '-' || \_str[i] == '\*' || \_str[i] == '/') {

double a = e.Pop();

double b = e.Pop();

e.Push(Calculator(b, a, \_str[i]));

}

else {

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

if(B[j] == \_str[i])

e.Push(VO[j]);

}

}

return e.Pop();

}

bool Converter::Priority(char a, char b) {

if ((a == '\*' || a == '/') && (b == '\*' || b == '/')) return true;

if ((a == '+' || a == '-') && (b == '+' || b == '-')) return true;

if ((a == '+' || a == '-') && (b == '\*' || b == '/')) return true;

return false;

}

double Converter::Calculator(double a, double b, char o) {

switch (o)

{

case '+':

return a + b;

case '-':

return a - b;

case '\*':

return a \* b;

case '/':

if (b == 0) throw "You cannot divide by zero";

return a / b;

}

}

void Converter::GetValueOperands(const string &\_str, double\* &VO, char\* &B, int &ki) {

ki--;

for (int i = 0; i < \_str.length(); i++)

if (\_str[i] != '\*' && \_str[i] != '/' && \_str[i] != '+' && \_str[i] != '-')

ki++;

VO = new double[ki];

B = new char[ki];

int c = 0;

int g;

for (int i = 0; i < \_str.length(); i++) {

g = 0;

if (\_str[i] != '\*' && \_str[i] != '/' && \_str[i] != '+' && \_str[i] != '-') {

for (int j = 0; j < c; j++) if (B[j] == \_str[i]) {

g = 1;

break;

}

if (g == 0) {

B[c] = \_str[i];

cout << "Enter " << \_str[i] << ": ";

cin >> VO[c];

c++;

}

}

}

}