Министерство Образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Нижегородский Государственный Университет им. Н.И. Лобачевского» «Институт Информационных Технологий Математики и Механики»

**Отчет**

**По лабораторной работе №3**

**«Вычисление арифметических выражений с помощью постфиксной формы записи»**

Выполнил:

Студент группы 0826-1

Смертин Д. С.

Проверил:

Старший преподаватель

Кафедры МОСТ

Сысоев А. В.

г. Нижний Новгород

2016 г.

Оглавление

[Введение 3](#_Toc468821597)

[Постановка задачи 4](#_Toc468821598)

[Руководство пользователя 5](#_Toc468821599)

[Руководство программиста 7](#_Toc468821600)

[Алгоритмы 9](#_Toc468821601)

[Заключение 10](#_Toc468821602)

[Приложение 11](#_Toc468821603)

# Введение

В лабораторной работе был реализован класс стека. Стек это динамическая структура данных, представляющая собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»). Стеки, в данной рабте, нужны для перевода инфиксного арифметического выражения в постфиксное. Пример: a+b\*c – это обычное (инфиксное) арифметическое выражение. Постфиксная форма записи данного выражения будет выглядеть так: a b c \* +. Как переводится арифметическое выражение из инфиксной формы в постфиксную будет описано далее. Также стек позволяет вычислять арифметические выражения с заданными параметрами a, b и c. Об этом так же будет рассказано позже.

# Постановка задачи

Цель данной лабораторной работы — разработка структуры данных Стек и ее использование для расчета арифметических выражений с использованием обратной польской записи (постфиксной формы).

В данной лабораторной работе нужно написать программу, которая должна выполнять следующие действия:

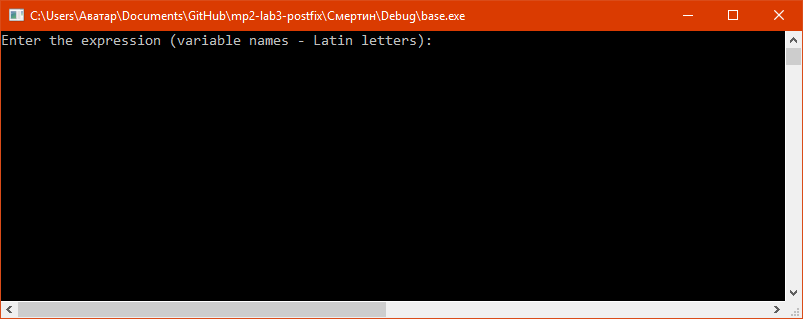
1. Проверять арифметические выражения на корректность;
2. Разбирать арифметические выражения на элементы;
3. Переводить арифметические выражения в постфиксную форму;
4. Заполнять таблицу переменных;
5. Вычислять арифметические выражения.

Для этого необходимо:

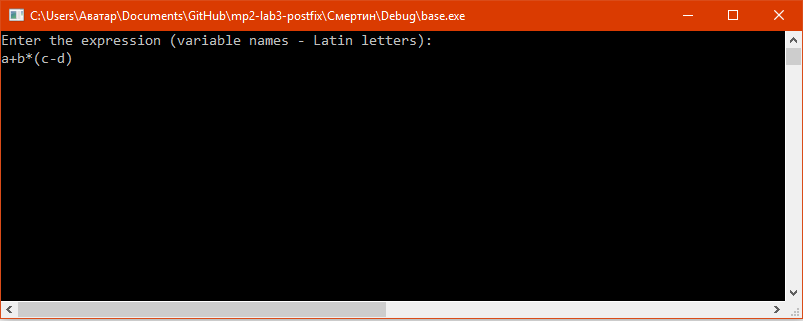
1. Разработать интерфейс шаблонного класса TStack.
2. Реализовать методы шаблонного класса TStack.
3. Разработать интерфейс класса TPostfix для работы с постфиксной формой.
4. Реализовать методы класса TPostfix.
5. Разработать и реализовать тесты для классов TStack и TPostfix на базе Google Test.

# Руководство пользователя

Для запуска программы перейдите в папку «mp2-lab3-postfix», далее перейдите в папку «Смертин», далее в папку «Debug». В данной папке найдите и запустите «base.exe». На экране появится консоль:

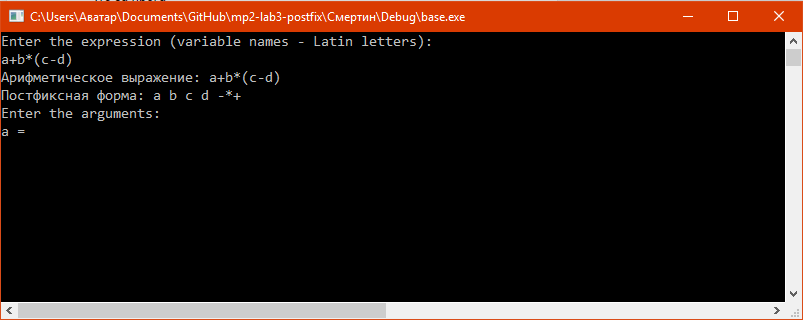


Далее нужно ввести арифметическое выражение. В Вашем выражении допускаются только заглавные и прописные латинские буквы, цифры, скобки, операции: «+», «-», «\*», «/» -- и знак «точка». Если будет введен некорректный символ, то программа завершит работу. Также выражение не должно начинаться и заканчиваться с арифметических операций, начинаться с закрывающейся скобки «)» и заканчиваться открывающейся скобкой «(». Следите за общим количеством скобок: количество открывающихся скобок и закрывающихся должно совпадать. В выражении должна отсутствовать комбинация арифметической операции и скобки, например «+)» или «(-», а также комбинации «()» и «)(». Следите, чтоб количество переменных соответствовало количеству операций. Иначе программа завершит работу.

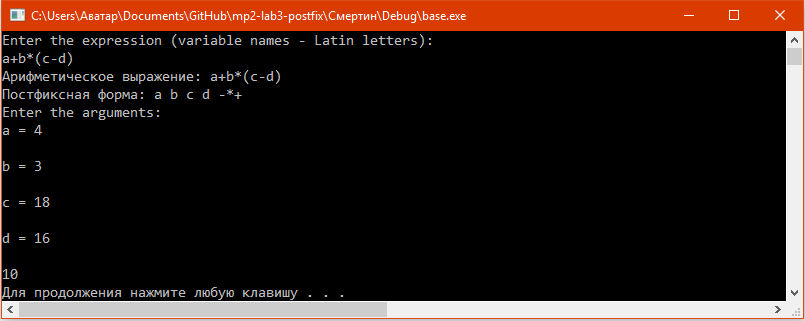


После ввода выражения, на экран оно выведется снова и выведется его постфиксная форма.

Далее нужно ввести числовые значения переменных:



Далее программа вычислит арифметическое выражение и выведет его на экран:



Для предотвращения ошибок вводите логически правильные арифметические выражения и значения переменных.

# Руководство программиста

Структура программы

Лабораторная работа состоит из 3 проектов: «base», «base\_test» и «gtest».

«base»

Проект «base» состоит из двух заголовочных файлов «stack.h» и «postfix.h», в которых реализованы соответственно класс-шаблон TStack и класс TPostfix. Из названий становится ясно, что в TStack реализован стек, а в TPostfix реализованы методы для работы с арифметическим выражением. Также в этом проекте есть файл «sample\_postfix.cpp», который содержит функцию «int main()».

«stack.h»

В класс-шаблоне TStack объявлены поля:

1. T \*pMem // -- сама структура стека, память для хранения данных
2. int size // -- размер стека
3. int top // -- индекс последнего элемента

Методы, объявленные в TStack:

1. TStack (int \_size) // -- конструктор, передающий размер стека
2. ~TStack() // -- деструктор
3. bool IsEmpty() // проверяет на пустоту стека
4. bool IsFull() // проверяет на полноту стека
5. void Put(T val) // добавляет элемент в стек
6. T Get() // извлекает элемент из стека
7. int GetTop() // возвращает индекс последнего элемента
8. int GetSize() // возвращает размер стека
9. T GetValTop() // возвращает значение последнего элемента в стеке.

«postfix.h»

«postfix.h» содержит класс TPostfix и структуру TablFunc. Подключены библиотеки: <string>, <iostream>, <sstream> -- и "stack.h".

Структура TablFunc -- это таблица арифметических операций и их свойств. Каждой операции присвоены приоритет и количество операндов.

В классе TPostfix объявлены поля:

1. string infix // -- содержит инфиксное арифметическое выражение
2. string postfix // -- содержит постфиксное арифметическое выражение
3. string\* variable // -- массив переменных
4. size\_t varSize // -- длина массива переменных
5. TablFunc functions // -- таблица операций
6. double Res // -- результат вычисления арифметического выражения

Методы, объявленные в TPostfix:

1. TPostfix(string inf) // -- конструктор, принимающий арифм. выражение
2. bool CheckChars() // -- проверка на допустимые символы
3. bool CheckAmount() // -- проверка на кол-во переменных
4. bool CheckBrackets() // -- проверка на кол-во скобок
5. bool CheckInfix() // -- общая проверка введенного арифм. выражения
6. void ArrVarible() // -- выделяет из арифм. выражения переменные
7. string ToPostfix() // -- переводит выражение в постфиксную форму
8. double Calculate() // -- вычисляет арифметическое выражение
9. string GetInfix() // -- возвращает инфиксное выражение
10. string GetPostfix() // -- возвращает постфиксное выражение
11. string\* GetArrVar() // -- возвращает массив переменных.

«base\_test»

«base\_test» содержит 3 файла: «test\_tpostfix.cpp», «test\_tstack.cpp» и «test\_main.cpp». «test\_tpostfix.cpp» и «test\_tstack.cpp» содержат тесты, проверяющие классы TPostfix и TStack на корректную работу. К файлам подключена библиотека <gtest.h>.

# Алгоритмы

Алгоритм перевода инфиксного АВ в постфиксное

Пусть дано следующее АВ: a+b\*(c-e)=. Будем читать его слева на право.

1. Если встретилась переменная, то её заносим в постфиксную форму (ПФ);
2. Если встретилась «(«, то заносим её в стек операций;
3. Если встретилась операция (т.о.), то повторять следующее:
   1. Если стек операций пустой, то (т.о) -> стек;
   2. Если нет, то сравниваем приоритет (т.о.) и верхней операции стека (в.о.с.);
      1. Если приоритет (т.о.)<=(в.о.с.), то (в.о.с.)->ПФ;
      2. Иначе (т.о.)->стек и нужно прекратить повторять;
   3. Если встретилась «)», то все операции в стеке до «(« переносим в ПФ;
   4. Если встретилось «=», то все операции в стеке переносим в ПФ;

Таким образом, ПФ=abce-\*+.

Алгоритм вычисления АВ по его ПФ

1. Сначала нужно запросить у пользователя значения переменных;
2. Идем по ПФ слева на право:
   1. Если встречается переменная, то заносим её в стек переменных.
   2. Если встречается операция, то выполняем её, извлекая из стека переменные. Переменная в вершине стека будет стоять после операции, а следующая переменная будет стоять перед операцией.

# Заключение

Во время выполнения лабораторной работы я реализовал класс стека и научился им пользоваться. Также научился преобразовывать инфиксное арифметическое выражение в постфиксное и вычислять их с помощью компьютера. Была реализована программа, вычисляющая простейшие арифметические выражения. К программе были написаны тесты, подтверждающие корректность её работы.

# Приложение

«postfix.h»:

#ifndef \_\_POSTFIX\_H\_\_

#define \_\_POSTFIX\_H\_\_

#include <string>

#include "stack.h"

#include <iostream>

#include <sstream>

using namespace std;

#define MaxSizeString 1000

struct TablFunc

{

char func[7];

int priority[7];

int amount;

TablFunc()

{

func[0] = ')'; amount = 2;

func[1] = '('; priority[1] = 0;

func[2] = '+'; priority[2] = 1;

func[3] = '-'; priority[3] = 1;

func[4] = '\*'; priority[4] = 2;

func[5] = '/'; priority[5] = 2;

func[6] = '='; priority[6] = 0;

}

};

class TPostfix

{

string infix;

string postfix;

string\* variable;

size\_t varSize;

TablFunc functions;

double Res;

public:

TPostfix(string inf)

{

infix = inf;

CheckInfix();

}

bool CheckChars() // Проверка на допустимые символы

{

string valval = "()\*/-+abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.";

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

{

if (valval.find(infix[i]) == std::string::npos)

{

throw "An invalid character was found";

}

}

return true;

}

bool CheckAmount() // Проверка соответствия кол-ва переменных кол-ву операций

{

string str = " " + infix + " ";

string arop = "-\*/+";

string var = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

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

if (arop.find(str[i]) >= 0 && arop.find(str[i]) <= str.length())

{

if ((var.find(str[i - 1]) >= 0 && var.find(str[i - 1]) <= var.length())

&& (var.find(str[i + 1]) >= 0 && var.find(str[i - 1]) <= var.length())

|| (str[i + 1] == '(' && str[i - 1] == ')'))

continue;

else

throw "the number of variables does not correspond to the numbers of operations";

}

return true;

}

bool CheckBrackets() // Проверка количества скобок

{

string temp = infix;

if (temp.find('(') == std::string::npos && temp.find(')') == std::string::npos)

return true;

if (temp.find("()") >= 0 && temp.find("()") <= temp.length())

{

throw "Detected is ''()''";

}

while (temp.find('(') != std::string::npos || temp.find(')') != std::string::npos)

{

if (temp.find('(') >= 0 && temp.find('(') <= temp.length())

if (temp.find(')') == std::string::npos)

{

throw "Incorrect number of brackets";

}

else

{

temp[temp.find('(')] = ' ';

temp[temp.find(')')] = ' ';

}

else throw "Incorrect number of brackets";

}

return true;

}

bool CheckInfix() // Общая проверка

{

CheckChars();

CheckAmount();

CheckBrackets();

if (infix[0] == ')' || infix[0] == '\*' || infix[0] == '/' || infix[0] == '-' || infix[0] == '+')

throw "First character is operations";

if (infix[infix.length() - 1] == '(' || infix[infix.length() - 1] == '\*' || infix[infix.length() - 1] == '/' || infix[infix.length() - 1] == '-' || infix[infix.length() - 1] == '+')

throw "Last character is operations";

return true;

}

void ArrVarible() // Выделение из строки переменных и добавление их в массив

{

string inf = infix;

size\_t size = 0;

CheckInfix();

inf = inf + "=";

for (int i = 0; i < inf.size(); i++)

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

size++;

variable = new string[size];

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

variable[i] = "";

varSize = size;

int k = 0;

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

{

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

inf[i] = ' ';

}

while (inf.find('(') != std::string::npos || inf.find(')') != std::string::npos)

{

if (inf.find(')') >= 0 && inf.find(')') <= inf.length())

inf.erase(inf.find(')'), 1);

if (inf.find('(') >= 0 && inf.find('(') <= inf.length())

inf.erase(inf.find('('), 1);

}

while (inf.length() != 0)

{

if (inf.find(' ') >= 0 && inf.find(' ') <= inf.length())

{

size\_t pos = inf.find(' ');

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

{

variable[k] += inf[j];

}

inf.erase(0, pos + 1);

k++;

}

}

}

string ToPostfix()

{

string inf = infix;

postfix = "";

CheckInfix();

ArrVarible();

inf = inf + "=";

TStack<char> operations(MaxSizeString);

int i = 0;

while (inf.length() != 0)

{

while (i < varSize)

{

if (inf.find(variable[i]) == 0)

{

postfix += variable[i] + ' ';

inf.erase(0, variable[i].length());

i++;

break;

}

else

break;

i++;

}

if (inf[0] == '(')

{

operations.Put('(');

inf.erase(0, 1);

}

if (inf[0] == functions.func[2] || inf[0] == functions.func[3] || inf[0] == functions.func[4] || inf[0] == functions.func[5])

{

while (true)

{

if (operations.IsEmpty())

{

operations.Put(inf[0]);

break;

}

int i = 2;

int j = 2;

while (i < 6)

{

if (inf[0] == functions.func[i])

break;

else

i++;

}

while (j < 6)

{

if (operations.GetValTop() == functions.func[j])

break;

else

j++;

}

if (functions.priority[i] <= functions.priority[j])

{

postfix += operations.Get();

}

else

{

operations.Put(inf[0]);

break;

}

}

inf.erase(0, 1);

}

if (inf[0] == ')')

{

while (operations.GetValTop() != '(')

postfix += operations.Get();

operations.Get();

inf.erase(0, 1);

}

if (inf[0] == '=')

{

while (operations.IsEmpty() != true)

postfix += operations.Get();

inf.erase(0, 1);

}

}

return postfix;

}

double Calculate() // Ввод переменных, вычисление по постфиксной форме

{

cout << "Enter the arguments:" << endl;

double\* var = new double[varSize];

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

{

string ce = "0123456789.";

int j = 0;

while (j < variable[i].length())

{

if (ce.find(variable[i][j]) != std::string::npos)

j++;

else

break;

}

if (j == variable[i].length())

{

std::stringstream temp(variable[i]);

temp >> var[i];

}

else

{

size\_t k = 0;

while (k < i && i != 0)

if (variable[i] == variable[k])

{

var[i] = var[k];

break;

}

else

k++;

if (k == i || i == 0)

{

cout << variable[i] << " = ";

cin >> var[i];

string c = "0123456789.";

if (c.find(var[i]) != std::string::npos)

throw "Permission incorrect value";

cout << endl;

}

else

continue;

}

}

string post = postfix;

for (int i = 0; i < post.size(); i++)

if (post[i] == ' ')

post.erase(i, 1);

double tempResult = 0;

double result = 0;

TStack<double> varStack(varSize);

int i = 0;

while (post.length() != 0)

{

while (i < varSize)

{

if (post.find(variable[i]) == 0)

{

varStack.Put(var[i]);

post.erase(0, variable[i].length());

i++;

break;

}

else

break;

}

if (post[0] == '\*')

{

double temp = varStack.Get();

tempResult = varStack.Get() \* temp;

varStack.Put(tempResult);

post.erase(0, 1);

}

if (post[0] == '/')

{

double temp = varStack.Get();

tempResult = varStack.Get() / temp;

varStack.Put(tempResult);

post.erase(0, 1);

}

if (post[0] == '+')

{

double temp = varStack.Get();

tempResult = varStack.Get() + temp;

varStack.Put(tempResult);

post.erase(0, 1);

}

if (post[0] == '-')

{

double temp = varStack.Get();

tempResult = varStack.Get() - temp;

varStack.Put(tempResult);

post.erase(0, 1);

}

}

result = varStack.Get();

return Res = result;

}

string GetInfix() { return infix; }

string GetPostfix() { return postfix; }

string\* GetArrVar() { return variable; }

};

#endif

«stack.h»:

#ifndef \_\_STACK\_H\_\_

#define \_\_STACK\_H\_\_

const int MaxStackSize = 1000;

template <class T>

class TStack

{

T \*pMem;

int size;

int top;

public:

TStack(int \_size)

{

size = \_size;

top = -1;

if ((size < 1) || (size > MaxStackSize))

throw size;

pMem = new T[size];

}

~TStack()

{

delete[] pMem;

}

bool IsEmpty()

{

return top == (-1);

}

bool IsFull()

{

return top == (size - 1);

}

void Put(T val)

{

if (IsFull())

throw val;

pMem[++top] = val;

}

T Get()

{

if (IsEmpty())

throw 0;

return pMem[top--];

}

int GetTop()

{

return top;

}

int GetSize()

{

return size;

}

T GetValTop()

{

return pMem[top];

}

};

#endif

«sample\_postfix.cpp»:

#include <iostream>

#include <string>

#include "postfix.h"

using namespace std;

int main()

{

double res;

string inf;

setlocale(LC\_ALL, "Russian");

cout << "Enter the expression (variable names - Latin letters):" << endl;

cin >> inf;

TPostfix postfix(inf);

cout << "Арифметическое выражение: " << postfix.GetInfix() << endl;

postfix.ToPostfix();

cout << "Постфиксная форма: " << postfix.GetPostfix() << endl;

res = postfix.Calculate();

cout << res << endl;

system("pause");

return 0;

}