МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Отчет по лабораторной работе**

на тему:

**«Постфиксная запись арифметического выражения»**

**Выполнил:** студент группы 3822Б1ПР2

Наумов Богдан Александрович

Нижний Новгород

2023

Оглавление

[1. Введение 3](#_Toc153708938)

[2. Постановка задачи 4](#_Toc153708939)

[3. Руководство пользователя 5](#_Toc153708940)

[4. Руководство программиста 6](#_Toc153708941)

[4.1.Описание структуры программы 6](#_Toc153708942)

[4.2.Описание структур данных 6](#_Toc153708943)

[4.3.Описание алгоритмов 9](#_Toc153708944)

[5. Эксперименты 10](#_Toc153708945)

[6. Заключение 11](#_Toc153708946)

[7. Приложение 12](#_Toc153708947)

## Введение

Постфиксная запись арифметического выражения (также известная как обратная польская запись) - это способ записи математических выражений, в котором операторы следуют после своих операндов. Например, выражение "3 + 4" будет записано как "3 4 +". Этот метод записи упрощает вычисление выражений, так как не требуется использование скобок или определения приоритета операций. Она часто используется в компьютерных научных вычислениях, а также в калькуляторах с обратной польской записью. Постфиксная запись также может быть полезна при реализации алгоритмов обработки выражений в программировании.

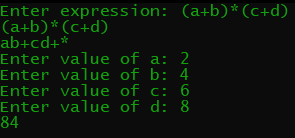
## Постановка задачи

Чтобы выполнить работу нужно решить следующие задачи:

1. Реализовать вспомогательный шаблонный класс TStack.
2. Реализовать шаблонный класс TArithmeticExpression.
3. Написать пример программы, которая будет демонстрировать работу класса TArithmeticExpression.
4. Написать набор тестов с использованием GoogleC++ TestingFramework для проверки работы классов TStack и TArithmeticExpression.

## Руководство пользователя

При запуске программы, демонстрирующей работу класса TArithmeticExpression, она просит ввести арифметическое выражение. После чего она просит ввести значения всех найденных в выражении операндов и далее выводит результат вычислений.



*Рис.1 (результат работы програаммы)*

## 4. Руководство программиста

## 4.1.Описание структуры программы

Программа состоит из следующих модулей:

* Модуль **include**. Статическая библиотека. Включает в себя заголовочные файлы stack.h и TArithmeticExpression.h, в которых описаны методы с реализацией классов TStack и TArithmeticExpression.
* Модуль **tests\_Stack**. Набор тестов для реализованных классов. Включает в себя файлы **test\_stack.cpp** и **test\_TArithmeticExpression.cpp**. Реализованы они с помощью использования фреймворка **Google Test**.
* Модули **sample\_TArithmeticExpression** и **sample\_stack** - примеры использования классов. Включает в себя файлы с реализациями **sample\_stack.cpp** и **sample\_TArithmeticExpression.cpp**.

## 4.2.Описание структур данных

Класс **TStack**:

Поля:

size\_t size – длина стека

T\* pMem – массив для хранения элементов стека

Конструкторы и деструктор:

TStack(size\_t \_size = 1)

TStack(TStack& st)

~TStack()

Методы:

bool IsFull() – проверка на заполненность стека

bool IsEmpty() – проверка на пустоту стека

void Push(T val) – положить элемент val в стек

T Pop() – возвращает верхний элемент стека

T Top() – возвращает верхний элемент стека, но не убирает его из стека

int GetSize() – возвращает размер стека

int GetTopInd() – возвращает индекс верхнего элемента стека

Перегрузки:

TStack& operator=(const TStack<T>& st) – присваивание

friend istream& operator>>(istream& istr, TStack& st) – ввод

friend ostream& operator<<(ostream& ostr, const TStack& st) – вывод

Класс **TArithmeticExpression**:

Поля:

string infix – инфиксная запись выражения

string postfix – постфиксная запись выражения

vector<char> lexems – вектор для хранения лексем выражения

map<char, double> operands – контейнер для хранения значений операндов выражения

map<char, int> priority - контейнер для хранения приоритетов операций

Конструктор:

TArithmeticExpression(const string& \_infix)

Методы:

void Parse() – разделяет инфиксную запись на лексемы

void ToPostfix() – переводит инфиксную запись в постфиксную

string GetInfix() const – возвращает инфиксную запись

string GetPostfix() const – возвращает постфиксную запись

vector<char> GetOperands() const – возвращает вектор операндов

double Calculate(const map<char, double>& values) – вычисляет значение выражения

## 4.3.Описание алгоритмов

Чтобы выполнить перевод арифметического выражения из инфиксной записи в постфиксную запись (обратную польскую запись), мы используем следующий алгоритм:

1. Создаем пустой стек для операторов.

2. Проходим по каждому символу в инфиксной записи (выражении) слева направо.

3 Если текущий символ является операндом (числом), мы добавляем его к нашей выходной строке (постфиксной записи).

4. Если текущий символ - открывающая скобка, помещаем ее в стек операторов.

5. Если текущий символ является закрывающей скобкой, мы извлекаем операторы из стека и добавляем их к нашей выходной строке, пока не встретим открывающую скобку. Затем мы удаляем открывающую скобку из стека.

6. Если текущий символ является оператором, мы извлекаем операторы из стека и добавляем их к нашей выходной строке до тех пор, пока приоритет текущего оператора не станет меньше приоритета оператора на вершине стека или до тех пор, пока стек не станет пустым. Затем мы помещаем текущий оператор в стек.

7. После обработки всех символов, мы извлекаем оставшиеся операторы из стека и добавляем их к нашей выходной строке.

## 5. Эксперименты

Проведем серию экспериментов, чтобы удостовериться в правильности работы алгоритма перевода.







## 6. Заключение

Все поставленные задачи в этой лабораторной работе были выполнены. Была разработана структура данных - стек и создан класс, который конвертирует инфиксное выражение в постфиксное и производит его вычисление. Основой арифметического выражения является стек. Благодаря самостоятельной работе на практике исследовалось устройство автоматических тестов. Это позволило проверить правильность работы реализованных классов. Также была проведена серия экспериментов для подтверждения правильности перевода.

## 7. Приложение

stack.h

#include <iostream>

using namespace std;

template <class T>

class TStack {

protected:

size\_t size;

int top;

T\* pMem;

public:

TStack(size\_t \_size = 1) : top(-1), size(\_size), pMem(new T[size]){

if (size <= 0)

throw - 1;

}

TStack(TStack& st) {

this->size = st.size;

this->pMem = new T[size];

this->top = st.top;

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

this->pMem[i] = st.pMem[i];

}

~TStack() {

if(this->pMem != nullptr)

delete[] this->pMem;

}

bool IsFull() {

return (top == size - 1);

}

bool IsEmpty() {

return top == -1;

}

void Push(T val) {

if (IsFull())

throw - 1;

pMem[++top] = val;

}

T Pop() {

if (IsEmpty())

throw - 1;

return pMem[top--];

}

T Top() {

if (IsEmpty())

throw - 1;

return pMem[top];

}

int GetSize() {

return this->size;

}

int GetTopInd() {

if (IsEmpty())

throw - 1;

return this->top;

}

TStack& operator=(const TStack<T>& st) {

if (this == &st)

return this;

if (this->size != st.size) {

if (this->pMem != nullptr) delete[] this->pMem;

this->size = st.size;

this->pMem = new T[size];

}

std::copy(st.pMem, st.pMem + st.size, this->pMem);

return \*this;

}

void print() {

for (int i = top; i >= 0; i--) {

if(i == top)

cout << pMem[i] << " <-top" << "\n";

else

cout << pMem[i] << "\n";

}

}

friend istream& operator>>(istream& istr, TStack& st) {

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

istr >> st.pMem[i];

return istr;

}

friend ostream& operator<<(ostream& ostr, const TStack& st)

{

for (size\_t i = st.top; i >= 0; i--) {

if (i == st.top)

ostr << st.pMem[i] << " <-top" << "\n";

else

ostr << st.pMem[i] << "\n";

}

return ostr;

}

};

TArithmeticExpression.h

#include <iostream>

#include <string>

#include <map>

#include <vector>

#include "stack.h"

using namespace std;

class TArithmeticExpression {

private:

string infix;

string postfix;

vector<char> lexems;

map<char, double> operands;

/\*static\*/ map<char, int> priority;

void Parse() {

for (char c : infix) {

if(c != ' ')

lexems.push\_back(c);

}

}

void ToPostfix() {

bool flag = false;

Parse();

TStack<char> st(20);

char stackItem;

for (char item : lexems) {

switch (item) {

case '(':

st.Push(item);

break;

case ')':

stackItem = st.Pop();

while (stackItem != '(') {

postfix += stackItem;

stackItem = st.Pop();

}

break;

case '+': case '-': case '\*': case '/':

flag = false;

if (st.IsEmpty())

st.Push(item);

else {

while (!st.IsEmpty()) {

stackItem = st.Top();

if (priority[item] <= priority[stackItem]) {

postfix += stackItem;

st.Pop();

}

else {

st.Push(item);

break;

}

if(st.IsEmpty()) flag = true;

}

if(flag) st.Push(item);

}

break;

default:

operands.insert({ item, 0.0 });

postfix += item;

}//end of switch

}//end of for

while (!st.IsEmpty()) {

stackItem = st.Pop();

postfix += stackItem;

}

}

public:

TArithmeticExpression(const string& \_infix) {

priority = { {'\*', 2}, {'/', 2}, {'+', 1}, {'-', 1}, {'(', 0}, {')', 0}};

infix = \_infix;

ToPostfix();

}

string GetInfix() const {

return this->infix;

}

string GetPostfix() const {

return this->postfix;

}

vector<char> GetOperands() const {

vector<char> op;

for (const auto& item : operands)

op.push\_back(item.first);

return op;

}

double Calculate(const map<char, double>& values) {

for (auto& val : values) {

try {

operands.at(val.first) = val.second;

}

catch(out\_of\_range &e){}

}

TStack<double> st(20);

double leftOperand, rightOperand;

for (char lexem : postfix) {

switch (lexem)

{

case '+':

rightOperand = st.Pop();

leftOperand = st.Pop();

st.Push(leftOperand + rightOperand);

break;

case '-':

rightOperand = st.Pop();

leftOperand = st.Pop();

st.Push(leftOperand - rightOperand);

break;

case '\*':

rightOperand = st.Pop();

leftOperand = st.Pop();

st.Push(leftOperand \* rightOperand);

break;

case '/':

rightOperand = st.Pop();

leftOperand = st.Pop();

st.Push(leftOperand / rightOperand);

break;

default:

st.Push(operands[lexem]);

break;

}

}

return st.Pop();

}

};

sample\_TArithmeticExpression.cpp

#include "TArithmeticExpression.h"

int main() {

string exprStr;

cout << "Enter expression: ";

cin >> exprStr;

TArithmeticExpression expr(exprStr);

cout << expr.GetInfix() << endl;

cout << expr.GetPostfix() << endl;

vector<char> operands = expr.GetOperands();

map<char, double> values;

double val;

for (const auto& op : operands) {

cout << "Enter value of " << op << ": ";

cin >> val;

values[op] = val;

}

cout << expr.Calculate(values) << endl;

}