МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

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**

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

**«Постфиксная форма записи арифметических выражений»**

**Выполнил(а):** студент группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Хохлов А.Д./

Подпись

**Проверил:** к.т.н, доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д./

Подпись

Нижний Новгород  
2023

Содержание

[Введение 3](#_Toc150891848)

[1 Постановка задачи 4](#_Toc150891849)

[2 Руководство пользователя 5](#_Toc150891850)

[2.1 Приложение для демонстрации работы векторов 5](#_Toc150891851)

[2.2 Приложение для демонстрации работы матриц 5](#_Toc150891852)

[3 Руководство программиста 7](#_Toc150891853)

[3.1 Описание алгоритмов 7](#_Toc150891854)

[3.1.1 Векторы 7](#_Toc150891855)

[3.1.2 Матрицы 9](#_Toc150891856)

[3.2 Описание программной реализации 11](#_Toc150891857)

[3.2.1 Описание класса TVec 11](#_Toc150891858)

[3.2.2 Описание класса TMatrix 15](#_Toc150891859)

[Заключение 18](#_Toc150891860)

[Литература 19](#_Toc150891861)

[Приложения 20](#_Toc150891862)

[Приложение А. Реализация класса TVec 20](#_Toc150891863)

[Приложение Б. Реализация класса TMatrix 22](#_Toc150891864)

# Введение

В современном мире информационных технологий большую роль играют арифметические операции. Одной из важных операций является эффективная работа с постфиксными формами записи. Постфиксная форма записи играет важную роль в многих областях информатики.

Знание и понимание структуры и принципов хранения помогают оптимизировать использование памяти и увеличивать эффективность вычислений.

Таким образом, данная лабораторная работа является актуальной и полезной для студентов и специалистов в области информационных технологий, которые имеют необходимость эффективно работать с битами и битовыми множествами.

# Постановка задачи

Цель: Целью данной лабораторной работы является создание структуры хранения и перевода инфиксной формы в постфиксную на языке программирования C++. В рамках работы необходимо разработать классы Stack, Lexems и Arifmetics, которые будут предоставлять функциональность для работы с инфиксной формой записи. Основной задачей является реализация основных операций с инфиксной формой записи в постфиксную.

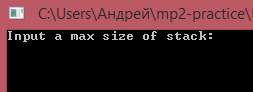
Задачи данной лабораторной работы:

* Разработка класса Stack.
* Реализация основных операций с стэком: push, pop, check, проверка на пустоту, очистка.
* Определение класса Lexems.
* Реализация основных операций с лексемами, включая разбиение на константы, переменные и операторы.
* Определение класса Arifmetics.
* Реализация основных операций для перевода инфиксной формы в постфиксную.
* Проверка и демонстрация работы разработанных классов с помощью приложений.
* Написание отчета о выполненной лабораторной работе, включая описание алгоритмов, программной реализации и результатов работы.

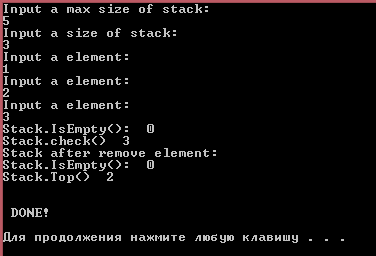
# Руководство пользователя

## Приложение для демонстрации работы стека

1. Запустите приложение с названием sample\_tstack.exe. В результате появится окно, показанное ниже (рис. 1).



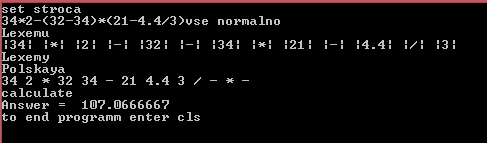
1. Основное окно программы
2. Это окно показывает работу основных функций работы со стеком (проверка на пустоту, получение верхнего элемента, удаление элемента). Для продолжения введите максимальный размер стека, количество элементов и сами элемета. В результате будет выведено (рис. 2). Для выхода нажмите любую клавишу.



1. Основное окно программы

## Приложение для демонстрации работы перевода в постфиксную форму

1. Запустите приложение с названием sample\_postfix\_form.exe. В результате появится окно, показанное ниже (рис. 3). Это окно показывает работу основных функций перевода инфиксной формы в постфиксную.



1. Основное окно программы

# Руководство программиста

## Описание алгоритмов

### Стек

Стек — это способ организации данных в компьютерной программе, при котором элементы добавляются и удаляются с одного конца (вершины стека), а доступ к элементам осуществляется только через другой конец (основание стека). Таким образом, стек работает по принципу "последним пришел — первым вышел" (LIFO).

**Операция добавления в конец:**

Операция добавляет новый элемент в вершину стека.

Пример:

V = {1, 2, 3, 4, 5}

Добавление нового элемента:

elem = 2

V + elem= {1, 2, 3, 4, 5, 2}

**Операция удаления элемента с верхушки стека:**

Операция удаляет последний элемент с верхушки стека.

Пример:

V = {1, 2, 3, 4, 5}

После удаления:

V = {1, 2, 3, 4}

**Операция проверки верхушки стека:**

Операция возвращает последний элемент

Пример:

V = {1, 2, 3, 4, 5}

Результат выполнения операции:

А = 5

**Операция проверки на пустоту:**

Операция показывает наличие элементов в стеке. Возвращает 1 если он пустой, в противном случае 0.

Пример:

V = {0, 1, 2, 3, 4}

Результат выполнения операции:

0

### Постфиксная форма

Постфиксная форма арифметического выражения — это формат записи математического выражения, в котором операторы стоят после всех операндов. Например, вместо записи "3 + 4" используется "3 4 +". Постфиксная форма позволяет более ясно выразить порядок выполнения операций, поскольку операторы указываются после всех операндов.

Алгоритм, о котором ты говоришь, принимает на вход строку, которая содержит арифметическое выражение, и хэш-таблицу, в которой хранятся операнды этого выражения. Затем алгоритм использует математические правила для определения приоритета операций: сначала он выполняет операции в скобках, затем умножение и деление, а затем сложение и вычитание.

Пример перевода инфиксной формы в постфиксную:

Инфиксная форма: А+(B+C)

Постфиксная форма: ABC++

**Операция получения инфиксной записи:**

Возвращает исходное выражение.

**Операция получения постфиксной записи:**

Алгоритм:

* Проходим исходную строку;
* При нахождении **числа**, заносим его в выходную строку;
* При нахождении **оператора**, заносим его в стек;
* Выталкиваем в выходную строку из стека все операторы, имеющие приоритет выше рассматриваемого;
* При нахождении открывающейся скобки, заносим её в стек;
* При нахождении закрывающей скобки, выталкиваем из стека все операторы до открывающейся скобки, а открывающуюся скобку удаляем из стека.

Пример: x - y / (5 \* z) + 10

Результат: x y 5 z \* / - 10 +

## Описание программной реализации

### Описание класса Stack

class Stack {

private:

ValType \* arr;

int SM;

int ind;

public:

Stack(int Size\_Max);

~Stack();

Stack(const Stack& a);

void push(const ValType a);

ValType pop();

ValType check();

bool IsEmpty();

int Size();

void Clear();

void Print();

int getInd();

int GetSizeMax();

};

Назначение: представление стека.

Поля:

SM – максимальный размер стека.

ind – текущее колличество элементов.

arr – память для представления стека.

Методы:

Stack(int Size\_Max)

Назначение: конструктор с параметром.

Входные параметры:

~Stack();

Назначение: деструктор.

Stack(const Stack& a);

Назначение: конструктор копирования.

Выходные параметры:

a – копируемый стек.

void push(const ValType a);

Назначение: добавление элемента в начало.

Выходные параметры:

а – новый элемент.

ValType pop();

Назначение: удаление элемента с верхушки стека.

ValType check();

Назначение: получение значения элемента с верхушки стека.

Выходные параметры:

Значение элемента.

bool IsEmpty();

Назначение: проверка стека на пустоту.

Выходные параметры:

Результат проверки.

int Size();

Назначение: получение текущего размера стека.

Выходные параметры:

Значение максимального размера стека.

void Clear();

Назначение: очистка стека.

void Print();

Назначение: вывод стека.

int getInd();

Назначение: получение значения текущего количества элементов.

Выходные параметры:

Значение текущего количества элементов.

int GetSizeMax();

Назначение: получение максимального размера стека.

Выходные параметры:

Значение максимального размера стека.

### Описание класса Arifmetics

class Arifmetics {

private:

const char priority[6] = { ')','(','+','-','\*','/' };

const int m\_priority[6] = { -1,0,1,1,2,2 };

bool error = false;

char onestring[1000];

Stack<int> a;

Stack <int> b;

Lexems \* c;

Lexems \* d;

public:

Lexems\* GetPolish() {

return d;

}

char\* GetOnestring();

int SetVariable();

double calculate();

int Num(int i);

Arifmetics();

void Lexem();

void Polsky();

void PrintString();

int getPriority(int z);

bool Check();

void PrintPolsky();

void PrintLexem();

void SetStroka();

void PushStroka(char\* a);

friend class Stack<int>;

};

Назначение: перевод арифметического выражения в постфиксную запись.

Поля:

const char priority[6] –операторы.

const int m\_priority[6] – приоритет операторов.

char onestring[1000] – строка записи.

Stack<int> a – стек для операторов.

Stack <int> b – стек для операторов.

Lexems \* c – массив лексем.

Lexems \* d – массив лексем.

Методы:

TMatrix(int mn = 10);

Назначение: конструктор с параметром.

Входные параметры:

mn – размер матрицы.

TMatrix(const TMatrix& mtrx);

Назначение: конструктор копирования.

Входные параметры:

mtrx – ссылка на константную матрицу.

TMatrix(const TVec<TVec<T>>& mtrx);

Назначение: конструктор преобразования типа.

Входные параметры:

mtrx – ссылка на константный вектор векторов.

const TMatrix operator=(const TMatrix& mtrx);

Назначение: перегрузка оператора присваивания.

Входные параметры:

mtrx – ссылка на константную матрицу.

Выходные параметры:

Константная результирующая матрица.

bool operator==(const TMatrix& mt)const;

Назначение: перегрузка оператора сравнения на равенство.

Входные параметры:

mt – ссылка на константную матрицу.

Выходные параметры:

Результат сравнения.

bool operator!=(const TMatrix& mt)const;

Назначение: перегрузка оператора сравнения на неравенство.

Входные параметры:

mt – ссылка на константную матрицу.

Выходные параметры:

Результат сравнения.

TMatrix operator+(const TMatrix& mt);

Назначение: перегрузка оператора суммы.

Входные параметры:

mt – матрица.

Выходные параметры:

Результирующая матрица.

TMatrix operator-(const TMatrix& mt);

Назначение: перегрузка оператора разности.

Входные параметры:

mt – матрица.

Выходные параметры:

Результирующая матрица.

TMatrix operator\*(const TMatrix& mt);

Назначение: перегрузка оператора произведения.

Входные параметры:

mt – матрица.

Выходные параметры:

Результирующая матрица.

friend istream& operator>>(istream& istr, TMatrix<T>& mt);

Назначение: перегрузка операции потокового ввода.

Входные параметры:

istr – ссылка на поток ввода.

mt – ссылка на матрицу.

Выходные параметры:

Ссылка на поток ввода.

friend ostream& operator<<(ostream& ostr, const TMatrix<T>& mt);

Назначение: перегрузка операции потокового вывода.

Входные параметры:

ostr – ссылка на поток вывода.

mt – ссылка на константную матрицу.

Выходные параметры:

Ссылка на поток вывода.

# Заключение

В ходе выполнения работы "Векторы и матрицы" были изучены и практически применены концепции верхнетреугольных матриц.

Были достигнуты следующие результаты:

1. Были изучены теоретические основы векторов и матриц

2. Была разработана программа, реализующая операции над векторами и матрицами. В ходе экспериментов была оценена эффективность работы этих операций и сравнена с другими подходами.

3. Были проанализированы полученные результаты и сделаны выводы о преимуществах и ограничениях использования векторов. Оказалось, что эти структуры данных особенно полезны при работе с большими объемами данных, где компактность представления и эффективность операций являются ключевыми факторами.

# Литература

1. Сысоев А.В., Алгоритмы и структуры данных, лекция 05, 5 октября.

# Приложения

## Приложение А. Реализация класса TVec

template <class T>

TVec<T>::TVec(int size, int start\_ind) {

if (size < 0)

throw "invalid size";

this->size = size;

if (start\_ind < 0)

throw "invalid start index";

this->start\_ind = start\_ind;

pMem = new T[size];

}

template <class T>

TVec<T>::TVec(const TVec<T>& vec) {

size = vec.size;

start\_ind = vec.start\_ind;

pMem = new T[size];

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

pMem[i] = vec.pMem[i];

}

template <class T>

TVec<T>::~TVec() {

delete[] pMem;

}

template <class T>

int TVec<T>::GetSize()const noexcept {

return size;

}

template <class T>

int TVec<T>::GetStartIndex()const noexcept {

return start\_ind;

}

template <class T>

T& TVec<T>::operator[](const int index) {

return pMem[index];

}

template <class T>

T& TVec<T>::operator[](const int index)const {

return pMem[index];

}

template <class T>

bool TVec<T>::operator==(const TVec<T>& vec)const {

if (start\_ind != vec.start\_ind)

return false;

if (size != vec.size)

return false;

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

if (pMem[i] != vec.pMem[i])

return false;

return true;

}

template <class T>

bool TVec<T>::operator!=(const TVec<T>& vec)const {

return !(\*this == vec);

}

template <class T>

TVec<T> TVec<T>::operator\*(const T& value) {

TVec<T> tmp(\*this);

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

tmp.pMem[i] = pMem[i] \* value;

}

return tmp;

}

template <class T>

T TVec<T>::operator\*(const TVec<T>& vec) {

if (start\_ind != vec.start\_ind)

throw "invalid size";

if (size != vec.size)

throw "invalid start index";

T res = T();

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

res += pMem[i] \* vec.pMem[i];

return res;

}

template <class T>

TVec<T> TVec<T>::operator+(const TVec<T>& vec) {

if (start\_ind != vec.start\_ind)

throw "invalid size";

if (size != vec.size)

throw "invalid start index";

TVec<T> tmp(size, start\_ind);

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

tmp.pMem[i] = pMem[i] + vec.pMem[i];

return tmp;

}

template <class T>

TVec<T> TVec<T>::operator-(const TVec<T>& vec) {

if (start\_ind != vec.start\_ind)

throw "invalid size";

if (size != vec.size)

throw "invalid start index";

TVec<T> tmp(size, start\_ind);

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

tmp.pMem[i] = pMem[i] - vec.pMem[i];

return tmp;

}

template <class T>

TVec<T> TVec<T>::operator+(const T& value) {

TVec<T> tmp(size, start\_ind);

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

tmp.pMem[i] = pMem[i] + value;

return tmp;

}

template <class T>

TVec<T> TVec<T>::operator-(const T& value) {

TVec<T> tmp(size, start\_ind);

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

tmp.pMem[i] = pMem[i] - value;

return tmp;

}

template <class T>

const TVec<T>& TVec<T>::operator=(const TVec<T>& vec) {

if (\*this == vec)

return \*this;

if (size != vec.size)

{

size = vec.size;

delete[] pMem;

pMem = new T[size];

}

start\_ind = vec.start\_ind;

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

pMem[i] = vec.pMem[i];

return \*this;

}

friend ostream& operator<<(ostream& ostr, const TVec<T>& vec)

{

ostr << "|";

for (int i = 0; i < vec.start\_ind; i++) {

ostr << "0 ";

}

for (int i = 0; i < vec.size - 1; i++) {

ostr << vec.pMem[i] << " ";

}

ostr << vec.pMem[vec.size-1];

ostr << "|" << endl;

return ostr;

}

friend istream& operator>>(istream& istr, TVec<T>& vec)

{

cout << endl << "vec(start ind = " << vec.GetStartIndex() << " size = " << vec.GetSize() << ") = ";

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

istr >> vec.pMem[i];

}

cout << endl;

return istr;

}

## Приложение Б. Реализация класса TMatrix

template <class T>

TMatrix<T>::TMatrix(int mn) : TVec<TVec<T>>(mn) {

if (mn < 0)

throw "invalid size";

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

pMem[i] = TVec<T>(mn - i, i);

}

template <class T>

TMatrix<T>::TMatrix(const TMatrix& mtrx) : TVec<TVec<T>>((TVec<TVec<T>>)mtrx) {}

template <class T>

TMatrix<T>::TMatrix(const TVec<TVec<T>>& mtrx) : TVec<TVec<T>>(mtrx) {}

template <class T>

const TMatrix<T> TMatrix<T>::operator=(const TMatrix& mtrx) {

return TVec<TVec<T>>::operator=(mtrx);

}

template <class T>

bool TMatrix<T>::operator==(const TMatrix& mt)const {

return TVec<TVec<T>> :: operator == (mt);

}

template <class T>

bool TMatrix<T>::operator!=(const TMatrix& mt)const {

return TVec<TVec<T>> :: operator != (mt);

}

template <class T>

TMatrix<T> TMatrix<T>::operator+(const TMatrix& mt) {

if (size != mt.size)

throw "invalid size";

return TVec<TVec<T>> :: operator + (mt);

}

template <class T>

TMatrix<T> TMatrix<T>::operator-(const TMatrix& mt) {

if (size != mt.size)

throw "invalid size";

return TVec<TVec<T>> :: operator - (mt);

}

template <class T>

TMatrix<T> TMatrix<T>::operator\*(const TMatrix& mt) {

if (size != mt.size)

throw "invalid size";

TMatrix res(size);

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

for (int j = i; j < size; ++j) {

res[i][j - i] = 0;

}

}

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

for (int j = i; j < size; ++j) {

for (int k = i; k <= j; ++k) {

res[i][j - i] += (\*this)[i][k - i] \* mt[k][j - k];

}

}

}

return res;

}

friend istream& operator>>(istream& istr, TMatrix<T>& mt)

{

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

istr >> mt.pMem[i];

}

return istr;

}

friend ostream& operator<<(ostream& ostr, const TMatrix<T>& mt)

{

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

ostr << mt.pMem[i];

}

return ostr;

}