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**

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

**«Стек»**

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Чижов М.А./

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#__RefHeading___1)

[1 Постановка задачи 5](#__RefHeading___2)

[2 Руководство пользователя 6](#__RefHeading___3)

[2.1 Приложение для демонстрации работы стека 6](#__RefHeading___4)

[3 Руководство программиста 7](#__RefHeading___5)

[3.1 Описание алгоритмов 7](#__RefHeading___6)

[3.1.1 Стек 7](#__RefHeading___7)

[3.2 Описание программной реализации 8](#__RefHeading___8)

[3.2.1 Описание класса TStack 8](#__RefHeading___9)

[3.2.2 Описание класса TMatrix 11](#__RefHeading___10)

[Заключение 14](#__RefHeading___11)

[Литература 15](#__RefHeading___12)

[Приложения 16](#__RefHeading___13)

[Приложение А. Реализация класса TStack 16](#__RefHeading___14)

[Приложение Б. Реализация класса TMatrix 17](#__RefHeading___15)

# Введение

Стек — это одна из основных структур данных в программировании. Он представляет собой упорядоченную коллекцию элементов, где добавление новых элементов и удаление существующих осуществляется в определенном порядке — «последним пришел, первым ушел». Такой принцип работы стека называется принципом LIFO (Last In, First Out) или «последним пришел, первым вышел».

Особенностью стека является то, что доступ к элементам коллекции осуществляется только через вершину стека. Это значит, что при добавлении нового элемента он становится на вершину стека, а при удалении — удаляется элемент, находящийся на вершине. Такая организация стека позволяет реализовывать различные алгоритмы работающие с ограниченным набором данных, где необходимы операции добавления и удаления элементов в определенном порядке.

Применение стека в программировании широко распространено и находит свое применение в различных задачах. Одним из основных применений стека является реализация алгоритмов и структур данных, таких как обходы графов, рекурсивные функции, обратная польская запись и многое другое.

Стек также используется для сохранения состояния программы во время выполнения. Например, когда функция вызывается, текущее состояние программы (адрес возврата, значения локальных переменных и т.д.) сохраняется в стеке. При завершении функции состояние восстанавливается из стека, чтобы выполнение программы продолжилось с того места, где оно было прервано.

Кроме того, стек может использоваться для обработки и хранения временных данных в программе. Например, стек может использоваться для реализации механизма отката действий в текстовом редакторе, где каждое действие добавляется в стек и может быть отменено путем извлечения последнего элемента из стека.

Также стек может быть полезен при работе с рекурсией. Каждый раз, когда функция вызывает саму себя, новые переменные и значения сохраняются в стеке, позволяя программе вести учет всех рекурсивных вызовов и правильно возвращаться обратно при завершении рекурсии.

Использование стека позволяет программистам решать различные задачи эффективно и удобно. Однако при неправильном использовании стека может возникнуть переполнение или недостаток памяти, поэтому важно использовать его правильно и осуществлять контроль за его состоянием.

# Постановка задачи

Цель – реализовать структуру данных стек.

Задачи:

1. Реализовать класс для работы со стеком.
2. Написать следующие операции для работы со стеком: изъятие с вершины, вставка на вершину, проверка последнего элемента, проверка на пустоту, проверка на полноту.
3. Добавить вспомогательные операции получения размера и элемента по индексу.
4. Написать следующие алгоритмы для работы со стеками: перевод в постфиксную форму, вычисление выражения, записанного в постфиксной форме, перемещение элементов из одного стека в другой.

# Руководство пользователя

## Приложение для демонстрации работы стека

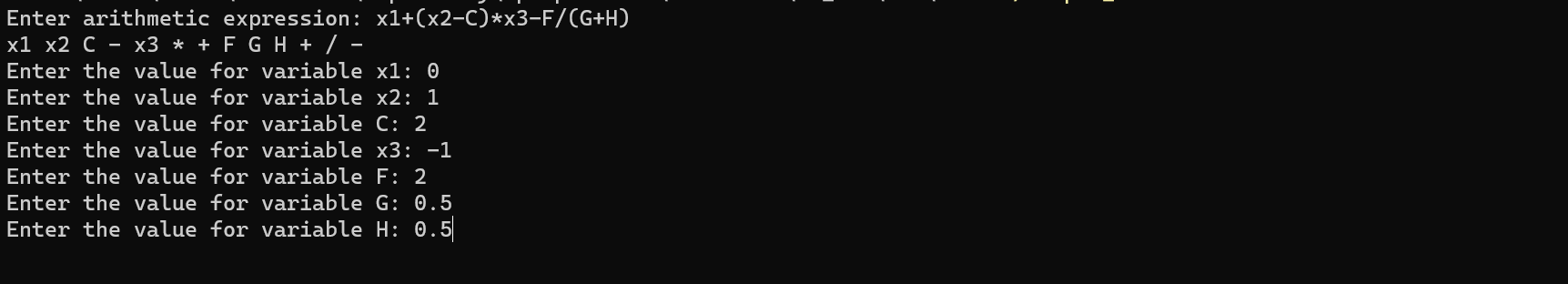
1. Запустите приложение с названием sample\_stack.exe. В результате появится окно, показанное ниже (рис. 1).



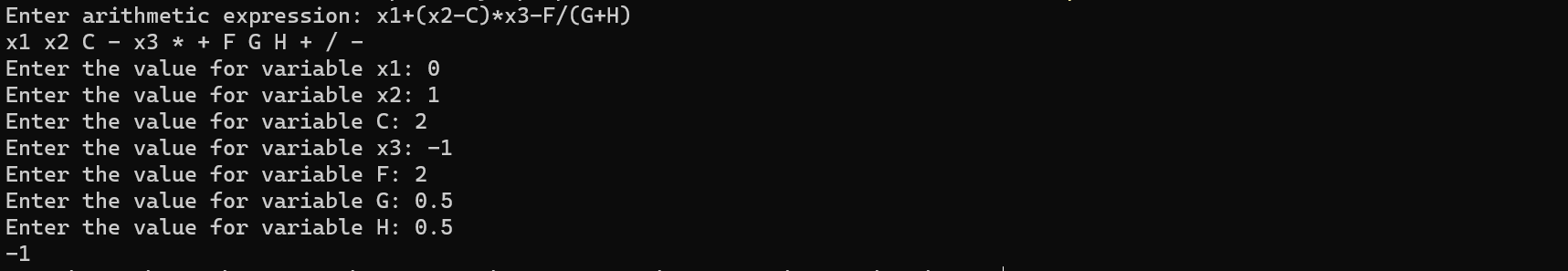
1. Основное окно программы
2. Введите арифметическое выражение. В результате выведется выражение в постфиксной форме (рис. 2).



1. Вывод постфиксной формы
2. Введите значения переменных (рис. 3).



1. Ввод значений переменных
2. После ввода значений появится результат арифметического выражения (рис. 4).



1. Вывод результата

# Руководство программиста

## Описание алгоритмов

### Стек

Класс TStack представляет собой шаблонный класс. Для создания экземпляра класса TStack необходимо указать его размер. По умолчанию размер устанавливается как 10. Класс TStack также поддерживает ряд операций, включая добавление на вершину, изъятие с вершины, проверка последнего элемента, проверка на пустоту и полноту. Также присутствуют операции проверки элемента по индексу и размера стека.

**Операция получения размера стека**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 1 | 7 |  |

Результат:

4 – длина стека.

**Операция получения элемента по индексу**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 1 | 7 | 2 |

Результат:

1 – элемент с индексом 2.

**Операция добавления в стек**

Для этого используется флаг top, в данный момент top=2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 1 |  |  |

Добавим элемент 7

Результат:

top=3, элемент в стеке

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 1 | 7 |  |

**Операция изъятия с вершины**

Для этого также воспользуемся флагом top=4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 1 | 7 | 2 |

Результат:

Флаг top теперь стал равен 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 1 | 7 | 2 |

**Операция проверки на пустоту**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 1 | 7 |  |

Результат:

false – стек не пуст

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Результат:

true – стек пуст

**Операция проверки на полноту**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 1 | 7 | 2 |

Результат:

true – стек полон

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 8 | 3 | 9 |  |

Результат:

false – стек не полон

## Описание программной реализации

### Описание класса TStack

template <class T>

class TStack

{

private:

int maxSize;

int top;

T\* elems;

public:

TStack(int maxSize = 10);

TStack(const TStack<T>& s);

~TStack();

TStack<T>& operator=(const TStack<T>& s);

bool IsEmpty(void) const;

bool IsFull(void) const;

void ResizeStack();

int Length() const { return top+1; }

T GetElement(int ind) const;

T Top() const;

void Push(const T& elem);

void Pop();

friend ostream& operator <<(ostream& out, const TStack& s)

{

for (int i = 0; i <= s.top; i++)

out << s.elems[i] << ' ';

return out;

};

};

Назначение: представление вектора.

Поля:

vec – указатель на массив типа **T**.

size – размер вектора.

start\_ind – стартовый индекс вектора.

Методы:

TStack(int maxSize)

Назначение: Конструктор класса, инициализирующий стек заданного размера.

Входные параметры:

maxSize - максимальный размер стека, заданный при инициализации.

Выходные параметры: отсутствуют.

TStack(const TStack& s)

Назначение: Конструктор копирования, создающий копию стека.

Входные параметры **s** – объект класса **TStack**, который нужно скопировать.

Выходные параметры: отсутствуют.

~TStack();

Назначение: освобождает выделенную память для стека.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

bool IsEmpty() const;

Назначение: Метод, проверяющий, пуст ли стек.

Входные параметры: отсутствуют.

Выходные параметры: true – пуст, false в противном случае.

bool IsFull() const;

Назначение: Метод, проверяющий, заполнен ли стек.

Входные параметры: отсутствуют.

Выходные параметры: true – заполнен, false в противном случае.

void ResizeStack();

Назначение: Метод, увеличивающий размер стека при его переполнении.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

const TStack<T>& operator=(const TStack<T>& s);

Назначение: присваивание одного стека другому.

Входные параметры: s - объект класса TStack, который нужно присвоить.

Выходные параметры: ссылка на текущий объект класса **TStack**.

T GetElement(int ind) const;

Назначение: Метод, возвращающий элемент по указанному индексу.

Входные параметры: ind - индекс элемента, который требуется получить.

Выходные параметры: элемент стека по указанному индексу.

T Top() const;

Назначение: Метод, возвращающий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: верхний элемент стека.

void Push(const T& elem);

Назначение: Метод, помещающий элемент на вершину стека.

Входные параметры: elem - элемент, который требуется добавить на вершину стека.

Выходные параметры: отсутствуют.

void Pop();

Назначение: Метод, удаляющий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

friend ostream& operator <<(ostream& out, const TStack& s);

Назначение: оператор вывода для класса **TStack**.

Входные параметры:

**ostr** – ссылка на объект типа **ostream**, который представляет выходной поток.

**s** – ссылка на объект типа **TStack,** который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

### Описание класса TMatrix

template <typename T>

class TMatrix : public TVector<TVector<T>>

{

public:

TMatrix(int mn = 10);

TMatrix(const TMatrix<T>& m);

TMatrix(const TVector<TVector<T>>& m);

TVector<T>& operator[](const int index);

const TMatrix<T> operator=(const TMatrix<T>& m);

int operator==(const TMatrix<T>& m) const;

int operator!=(const TMatrix<T>& m) const;

TMatrix operator +(const TMatrix<T>& m);

TMatrix operator -(const TMatrix<T>& m);

TMatrix operator \*(const TMatrix<T>& m);

friend istream& operator >>(istream& in, TMatrix<T>& m)

{

for (int i = 0; i < m.GetSize(); i++)

in >> m.vec[i];

return in;

};

friend ostream& operator <<(ostream& out, TMatrix<T>& m)

{

for (int i = 0; i < m.GetSize(); i++)

out << m.vec[i] << endl;

return out;

};

};

Назначение: представление верхнетреугольной матрицы

Методы:

TMatrix(int mp);

Назначение: создает объект **TMatrix** заданного размера.

Входные параметры:  **mp** – размер матрицы.

Выходные параметры: созданный объект **TMatrix.**

TMatrix(const TMatrix<T>& m);

Назначение: создает копию объекта **TMatrix**.

Входные параметры: **m** - объект **TMatrix**, который нужно скопировать.

Выходные параметры: отсутствуют.

TMatrix(const TVector<TVector<T>>& m);

Назначение: создает объект **TMatrix** на основе векторов.

Входные параметры: **m** – ссылка на объект **TMatrix**.

Выходные параметры: созданный объект **TMatrix.**

TVector<T>& operator[](const int index);

Назначение: позволяет получить доступ к элементу матрицы по его индексу.

Входные параметры: **index** – индекс объекта.

Выходные параметры: элемент вектора типа **TVector**.

int operator==(const TMatrix<T>& m) const;

Назначение: сравнивает две матрицы на равенство.

Входные параметры: **m** – ссылка на объект **TMatrix**.

Выходные параметры: 1 – равны, 0 – не равны.

int operator!=(const TMatrix<T>& m) const;

Назначение: сравнивает две матрицы на неравенство

Входные параметры: **m** – ссылка на объект **TMatrix**.

Выходные параметры: 0 – равны, 1 – не равны.

const TMatrix<T> operator=(const TMatrix<T>& m);

Назначение: присваивание матрицы.

Входные параметры: **m** – ссылка на объект **TMatrix**.

Выходные параметры: ссылка на текущий объект класса **TMatrix**.

TMatrix operator +(const TMatrix<T>& m);

Назначение: сложение матриц.

Входные параметры: **m** – ссылка на объект **TMatrix**.

Выходные параметры: новый объект **TMatrix,** являющийсярезультатом сложения двух матриц.

TMatrix operator -(const TMatrix<T>& m);

Назначение: вычитание матриц.

Входные параметры: **m** – ссылка на объект **TMatrix**.

Выходные параметры: новый объект **TMatrix,** являющийсярезультатом вычитания двух матриц.

TMatrix operator \*(const TMatrix<T>& m);

Назначение: умножение двух матриц.

Входные параметры: **m** – ссылка на объект **TMatrix**.

Выходные параметры: новый объект **TMatrix,** являющийсярезультатом умножения двух матриц.

friend istream& operator >>(istream& in, TMatrix<T>& m);

Назначение: оператор ввода для класса **TMatrix**.

Входные параметры:

**in** – ссылка на объект типа **istream**, который представляет входной поток.

**m**– ссылка на объект типа **TMatrix**, в который будут считываться значения.

Выходные параметры: ссылка на объект типа **istream**.

friend ostream& operator <<(ostream& out, TMatrix<T>& m);

Назначение: оператор вывода для класса **TMatrix**.

Входные параметры:

**out** – ссылка на объект типа **ostream**, который представляет выходной поток.

**m** – ссылка на объект типа **TMatrix**, который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

# Заключение

В результате разработки классов TVector и TMatrix была создана структура данных, позволяющая эффективно работать с векторами и матрицами.

В результате разработки класса вектора мы получили возможность эффективно оперировать векторами и выполнять различные операции над ними, включая сложение, вычитание, умножение на скаляр, нахождение скалярного произведения, сложение и вычитание векторов, сравнение на равенство и неравенство. Класс вектора позволяет удобно хранить и обрабатывать числовые значения вектора и предоставляет удобные методы для доступа и модификации его элементов.

Также, благодаря разработке класса матрицы, мы получили возможность оперировать не только с векторами, но и с матрицами. Класс матрицы позволяет хранить и обрабатывать многомерную информацию и предоставляет методы для выполнения различных операций, включая сложение матриц, вычитание, умножение матриц и сравнение на равенство и неравенство.

# Литература

1. Что такое стек и как его использовать в программировании [https://podarkiyarki.ru/chto-takoe-stek-v-programmirovanii-i-kak-ego-ispolzovat]
2. Векторы и матрицы c++ [https://qaa-engineer.ru/vektory-i-matriczy-c/]

# Приложения

## Приложение А. Реализация класса TStack

template <class T>

TStack<T>::TStack<T>(int maxSize)

{

if (maxSize <= 0)

{

string msg = "Error";

throw msg;

}

this->maxSize = maxSize;

top = -1;

elems = new T[maxSize];

}

template <class T>

TStack<T>::TStack<T>(const TStack<T>& s)

{

maxSize = s.maxSize;

top = s.top;

elems = new T[maxSize];

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

elems[i] = s.elems[i];

}

template <class T>

TStack<T>::~TStack()

{

delete[] elems;

}

template <class T>

const TStack<T>& TStack<T>::operator=(const TStack<T>& s)

{

if (this == &s) {

return \*this;

}

delete[] elems;

maxSize = s.maxSize;

top = s.top;

elems = new T[maxSize];

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

elems[i] = s.elems[i];

}

return \*this;

}

template <class T>

bool TStack<T>::IsEmpty(void) const

{

return (top == -1);

}

template <class T>

bool TStack<T>::IsFull(void) const

{

if (maxSize - 1 == top)

return true;

return false;

}

template <class T>

void TStack<T>::ResizeStack()

{

int newMaxSize = maxSize \* 5; // Увеличиваем размер вдвое

T\* newElems = new T[newMaxSize]; // Создаем новый массив с увеличенным размером

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

newElems[i] = elems[i]; // Копируем элементы из старого массива в новый

delete[] elems; // Удаляем старый массив

elems = newElems; // Указываем на новый массив

maxSize = newMaxSize; // Обновляем значение maxSize

}

template <class T>

T TStack<T>::GetElement(int ind) const

{

if (ind < 0 || ind > top)

throw "error";

return elems[ind];

}

template <class T>

T TStack<T>::Top() const

{

if (top == -1)

{

string msg = "Error: stack is empty";

throw msg;

}

return elems[top];

}

template <class T>

void TStack<T>::Push(const T& elem)

{

if (IsFull())

ResizeStack();

elems[++top] = elem;

}

template <class T>

void TStack<T>::Pop()

{

if (IsEmpty())

{

string msg = "Error: stack is empty";

throw msg;

}

top -= 1;

}

## Приложение Б. Реализация класса TMatrix

template <typename T>

TMatrix<T> :: TMatrix(int mn) : TVector<TVector<T>> (mn)

{

if (mn <= 0)

throw "ERROR: matrix size less than 1";

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

(\*this)[i] = TVector<T>(mn - i, i);

}

template <typename T>

TMatrix<T>::TMatrix(const TMatrix<T>& m) : TVector<TVector<T>>(m) {};

template <typename T>

TMatrix<T>::TMatrix(const TVector<TVector<T>>& m) : TVector<TVector<T>>(m) {};

template <typename T>

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

return TVector<TVector<T>>::operator[](index);

}

template <typename T>

const TMatrix<T> TMatrix<T>::operator=(const TMatrix<T>& m)

{

return TVector<TVector<T>>::operator=(m);

};

template <typename T>

int TMatrix<T>::operator==(const TMatrix<T>& m) const

{

return TVector<TVector<T>>::operator==(m);

};

template <typename T>

int TMatrix<T>::operator!=(const TMatrix<T>& m) const

{

return !(\*this == m);

};

template <typename T>

TMatrix<T> TMatrix<T>::operator+(const TMatrix<T>& m)

{

return TVector<TVector<T>>::operator+(m);

};

template <typename T>

TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& m)

{

return TVector<TVector<T>>::operator-(m);

};

template <typename T>

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

{

if (TVector<TVector<T>>::GetSize() != m.GetSize() || TVector<TVector<T>>::GetStart() != m.GetStart())

throw "ERROR: matrix sizes do not match";

TMatrix<T> result(TVector<TVector<T>>::GetSize());

for (int i = 0; i < TVector<TVector<T>>::GetSize(); i++)

for (int j = result[i].GetStart(); j < TVector<TVector<T>>::GetSize(); j++)

result[i][j] = T(NULL);

for (int i = 0; i < TVector<TVector<T>>::GetSize(); i++)

{

for (int j = result[i].GetStart(); j < TVector<TVector<T>>::GetSize(); j++)

{

T sum = 0;

for (int k = result[i].GetStart(); k <= j; k++) {

sum += (\*this)[i][k] \* m.vec[k][j];

}

result.vec[i][j] = sum;

}

}

return result;

};

friend istream& operator >>(istream& in, TMatrix<T>& m)

{

for (int i = 0; i < m.GetSize(); i++)

in >> m.vec[i];

return in;

};

friend ostream& operator <<(ostream& out, TMatrix<T>& m)

{

for (int i = 0; i < m.GetSize(); i++)

out << m.vec[i] << endl;

return out;

};