МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
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**

Направление подготовки: «ПМИ»

**ОТЧЕТ**

по лабораторной работе

**ВЕКТОР**

**Выполнил:** студент группы

3822Б1ПМ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Р.Д. Махнёв

Подпись

**Проверил:** лаборант ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Черных

Подпись

Нижний Новгород  
2023

**Содержание**

1. [Введение 3](#_Toc132667869)
2. [Постановка задачи 3](#_Toc132667870)
3. [Описание алгоритмов 4](#_Toc132667871)
   1. [Полякласса вектор 4](#_Toc132667872)
   2. [Методы 4](#_Toc132667873)
4. [Описание программы 5](#_Toc132667874)
5. [Результаты 10](#_Toc132667875)
6. [Заключение 11](#_Toc132667876)
7. [Приложения 13](#_Toc132667877)

# **Введение**

**Актуальность.** Вектор -это стандартный шаблон обобщённого программирования языка C++, реализующий динамический массив. [2] Векторы в C++ представляют собой контейнеры, которые могут хранить объекты любого типа. Они позволяют добавлять и удалять элементы в конец контейнера, а также получать доступ к элементам по индексу. [1]

Векторы в C++ могут быть использованы для хранения большого количества данных и обеспечивают быстрый доступ к элементам.

**Цель работы:** изучение работы вектора

**Постановка задачи**

1. изучить различные источники по данной теме;
2. реализовать класс вектор;
3. провести сравнение различных методов сортировки.

**Описание алгоритмов**

1. **Поля класса вектор**
2. Указатель на область памяти, где будет находиться массив (\_array).
3. Размер вектора (count)
4. **Методы**
5. Обращение к элементу по индексу (at, operator[])
6. Получение размера вектора (getSize)
7. Обращение к первому и последнему элементу (getBack, getFront)
8. Добавление элемента в конец (push\_back)

Увеличивает размер массива и последнему

1. Изменение размера (resize)
2. Операции сравнения (==, !=, <, >, <=, >=)
3. Поэлементное сложение, вычитание, умножение, деление на скаляр, скалярное произведение. (operator+, operator-, operator\*, operator/)
4. Ввод, вывод (operator<<, operator>>)

(Приложение №3)

**Описание программы**

Данная программа является консольным приложением, она может выполнять операции с векторами.

Для примера создан целочисленный вектор (vector), его заполняют числами от 0 до 4.

Вызывается конструктор копирования из vectorв temp.

Выводится vector.

К tempприбавляется vector.

Вывод temp, скалярного произведенияtempна vector.

Вывод temp \* 3, temp \*= 3, temp /= 3.

Создан вектор manual длины 5, ввод manualиз консоли, и вывод.

Если массивы manualи tempсовпадают, то выводится You are lucky, в противном случае You are not lucky.

Если manual<temp, то выводится <

Если manual>temp, то выводится >

Если manual<= temp, то выводится <=

Если manual>= temp, то выводится >=

Вывод temp++ и ++temp.

Вызов расширения и сужения temp.

Вывод полученного вектора.

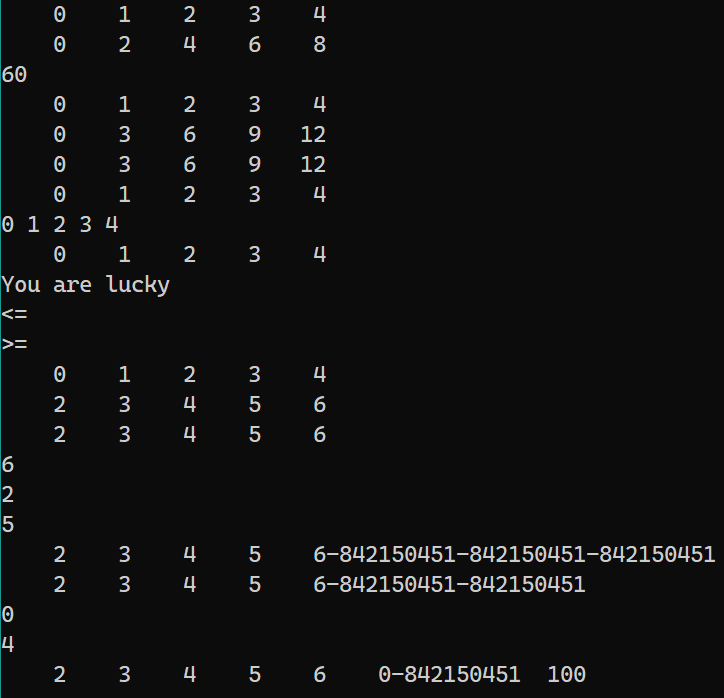
Запись по 5 индексу 0.

Вывод 5 элемента вектора.

Вывод 2 элемента вектора.

Запись в конец temp100.

Вывод полученного вектора. (Приложение №1)



**Описание классов**

**classVector–** содержит описание вектора

T\* \_array; - массив элементов вектора.

size\_tcount; - количество элементов в векторе.

Конструкторы:

Vector();

Vector(size\_t count);

Vector(const Vector<T>& \_vector);

Перегрузки операторов:

const bool operator==(const Vector<T>& \_vector) const;

Входные параметры: константная ссылка на вектор(\_vector).Результат – результат сравнения bool.

const bool operator!=(const Vector<T>& \_vector) const;

Входные параметры: константная ссылка на вектор(\_vector). Результат – результат сравнения bool.

Vector<T>&operator-= (constVector<T>& \_vector);

Входные параметры: константная ссылка на вектор(\_vector). Результат – ссылка на текущий вектор.

Vector<T>&operator=(constVector<T>& \_vector);

Входные параметры: константная ссылка на вектор(\_vector). Результат – ссылка на текущий вектор.

const bool operator<= (const Vector<T>& \_vector) const;

Входные параметры: константная ссылка на вектор (\_vector). Результат – результат сравнения bool.

const bool operator>= (const Vector<T>& \_vector) const;

Входные параметры: константная ссылка на вектор (\_vector). Результат – результат сравнения bool.

const bool operator< (const Vector<T>& \_vector) const;

Входные параметры: константная ссылка на вектор (\_vector). Результат – результат сравнения bool.

const bool operator> (const Vector<T>& \_vector) const;

Входные параметры: константная ссылка на вектор (\_vector). Результат – результат сравнения bool.

const Vector<T> operator+ (const Vector<T>& \_vector) const;

Входные параметры: константная ссылка на вектор (\_vector). Результат – вектор суммы.

const Vector<T> operator\* (const T& scalar) const;

Входные параметры: константная ссылка на скаляр (scalar). Результат – вектор произведения.

const Vector<T> operator- (const Vector<T>& \_vector) const;

Входные параметры: константная ссылка на вектор (\_vector). Результат – вектор разности.

const Vector<T> operator/ (const T& scalar) const;

Входные параметры: константная ссылка на скаляр (scalar). Результат – вектор частного.

Vector<T>& operator\*= (const T& scalar);

Входные параметры: константная ссылка на вектор (\_vector). Результат – ссылка на текущий вектор.

Vector<T>& operator+= (const Vector<T>& \_vector);

Входные параметры: константная ссылка на вектор (\_vector). Результат – ссылка на текущий вектор.

Vector<T>& operator/= (const T& scalar);

Входные параметры: константная ссылка на вектор (\_vector). Результат – ссылка на текущий вектор.

const Vector<T>& operator++();

Результат – ссылка на текущий вектор.

const Vector<T> operator++(int);

Результат – вектор до прибавления единицы ко всем элементам.

const T operator\* (const Vector<T>) const;

Входные параметры: константная ссылка на вектор (\_vector). Результат – скалярное произведения типа T.

T&operator[] (size\_tindex);

Входные параметры: индекс элемента (index). Результат – ссылка на элемент массива.

friend std::ostream& operator<<<T>(std::ostream& stream, const Vector<T>& \_vector);

Входные параметры: ссылка на ostream (stream), константная ссылка на вектор (\_vector). Результат – ссылка на stream.

friend const Vector<T> operator/ (const T& scalar, const Vector<T>& vector);

Входные параметры: константная ссылка на скаляр (scalar) T константная ссылка на вектор (\_vector). Результат – ссылка на текущий вектор.

friend std::istream& operator>><T> (std::istream& stream, const Vector<T>& \_vector);

Входные параметры: ссылка на istream (stream), константная ссылка на вектор (\_vector). Результат – ссылка наstream.

Методы:

Void resize(size\_tsize); - изменяет размер вектора.

T&getBack(); - возвращает ссылку на последний элемент вектора.

T&getFront(); - возвращает ссылку на первый элемент вектора.

voidpush\_back(constT&element); - добавляет элемент в конец списка.

size\_tgetSize(); - возвращает размер вектора.

T&at(size\_tindex);- возвращает ссылку на элемент вектора, с проверкой.

(Приложение №3)

**classException**– содержит описание исключения.

std::string error\_name; - имяо шибки.

std::string file\_name; - имя файла.

std::stringfunction; - имя функции, в которой произошла ошибка.

std::stringdate; - дата возникновения ошибки.

size\_tline; - строка, в которой возникла ошибка.

std::stringtime; - время возникновения ошибки.

Конструкторы:

Exception(const std::string&error\_name, const std::string&file\_name, const std::string& function, size\_t line)

Перегрузки операторов:

std::ostream& operator<< (std::ostream& stream, const Exception& exception)

Входныепараметры: ссылка на ostream (stream), константная ссылка на исключение (exception). Результат – ссылка наstream.

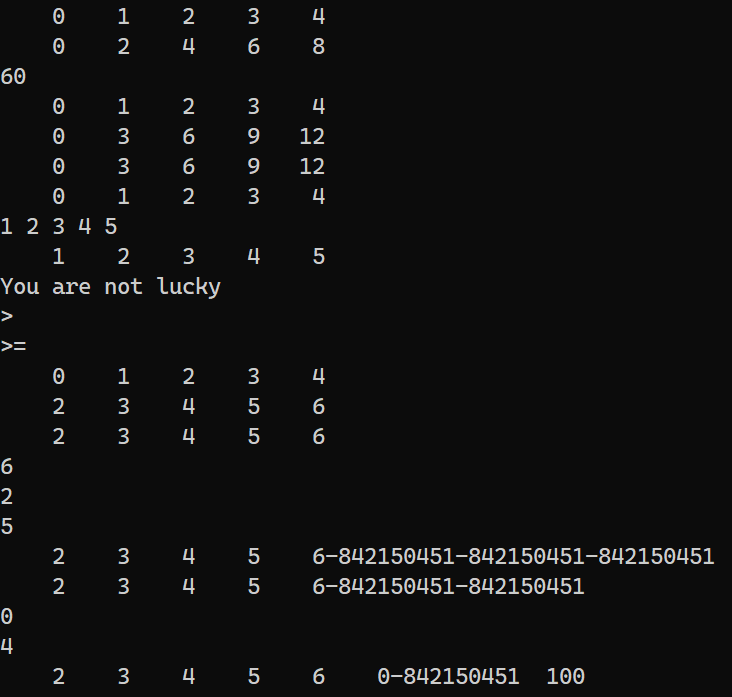
(Приложение №2)

**Описание функций:**

Main – главная функция. (Приложение №1)

**Результаты**

Результатом программы являются различные действия с векторами, сложение, вычитание, умножение на скаляр, скалярное произведение, деление на скаляр, ввод, вывод.



**Заключение**

В этой работе были изучен и разработан динамический массив в C++. По полученным данным можно сделать следующие выводы:

Программа выполняет поставленные задачи.

**Литература**

1. **C++ | Вектор – METANIT**

<https://metanit.com/cpp/tutorial/7.2.php>

1. **Vector (C++) — Википедия**

<https://ru.wikipedia.org/wiki/Vector_(C%2B%2B)>

**Приложения**

Приложение №1

#include"Vector.hpp"

#include"iostream"

intmain() try {

Vector<int>vector(5);

for (size\_ti = 0; i< 5; i++)

{

vector[i] = i;

}

Vector<int> temp = vector;

std::cout<< vector << std::endl;

temp += vector;

std::cout<< temp << std::endl;

std::cout<< (temp \* vector) << std::endl;

std::cout<< (temp -= vector) << std::endl;

std::cout<< (3 \* temp) << std:: endl;

std::cout<< (temp \*= 3) << std::endl;

std::cout<< (temp /= 3) << std::endl;

Vector<int>manual(5);

std::cin>> manual;

std::cout<< manual << std::endl;

if (manual == temp) {

std::cout<<"You are lucky"<< std::endl;

}

if (manual != temp) {

std::cout<<"You are not lucky"<< std::endl;

}

if (manual < temp) std::cout<<'<'<< std::endl;

if (manual <= temp) std::cout<<"<="<< std::endl;

if (manual > temp) std::cout<<'>'<< std::endl;

if (manual >= temp) std::cout<<">="<< std::endl;

std::cout<< temp++<< std::endl;

std::cout<<++temp << std::endl;

std::cout<< temp << std::endl;

std::cout<<temp.getBack() << std::endl;

std::cout<<temp.getFront() << std::endl;

std::cout<<temp.getSize() << std::endl;

temp.resize(8);

std::cout<< temp << std::endl;

temp.resize(7);

std::cout<< temp << std::endl;

temp.at(5) = 0;

std::cout<< temp.at(5) << std::endl;

std::cout<< temp[2]<< std::endl;

temp.push\_back(100);

std::cout<< temp << std::endl;

}

catch (Exception error) {

std::cout<< error;

}

Приложение №2

#pragmaonce

#include<ctime>

#include<string>

classException {

public:

std::stringerror\_name;

std::stringfile\_name;

std::string function;

std::string date;

size\_t line;

std::string time;

Exception(const std::string&error\_name, const std::string&file\_name, const std::string&function, size\_tline) {

this->error\_name=error\_name;

this->file\_name=file\_name;

this->function =function;

this->line = line;

this->time =\_\_TIME\_\_;

this->date =\_\_DATE\_\_;

}

};

std::ostream&operator<< (std::ostream&stream, constException&exception) {

stream<<"Error name: "<<exception.error\_name<< std::endl<<

"File: "<<exception.file\_name<< std::endl<<

"Function: "<<exception.function<< std::endl<<

"Line: "<<exception.line<< std::endl<<

"Date: "<<exception.date<< std::endl<<

"Time: "<<exception.time<< std::endl;

returnstream;

}

classOutOfRangeException :publicException {

public:

OutOfRangeException(const std::string&file\_name, const std::string&function, size\_tline) :Exception(std::string("OutOfRange"), file\_name, function, line) {}

};

Приложение №3

#pragmaonce

#include<ostream>

#include<istream>

#include<iostream>

#include<iomanip>

#include"Exception.hpp"

#defineSETW 5

template<typenameT1>

classVector;

template<typenameT>

std::ostream&operator<< (std::ostream&stream, constVector<T>&\_vector);

template<typenameT>

std::istream&operator>> (std::istream&stream, constVector<T>&\_vector);

template<typenameT>

classVector {

private:

T\* \_array;

size\_t count;

public:

Vector() {

\_array = nullptr;

count = 0;

}

Vector(size\_tcount) {

\_array = newT[count];

this->count = count;

}

Vector(constVector<T>&\_vector);

size\_tgetSize() {

return count;

}

T&at(size\_tindex) {

if (index>= count)

throwOutOfRangeException(\_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_);

return \_array[index];

}

~Vector() {

if (\_array != nullptr)

delete[] \_array;

count = 0;

}

voidpush\_back(constT&element);

T&getBack() {

return \_array[count - 1];

}

T&getFront() {

return \_array[0];

}

constbooloperator==(constVector<T>&\_vector) const;

voidresize(size\_tsize);

constbooloperator!=(constVector<T>&\_vector) const;

Vector<T>&operator-= (constVector<T>&\_vector);

Vector<T>&operator=(constVector<T>&\_vector);

constbooloperator<= (constVector<T>&\_vector) const;

constbooloperator>= (constVector<T>&\_vector) const;

constbooloperator< (constVector<T>&\_vector) const;

constbooloperator> (constVector<T>&\_vector) const;

constVector<T>operator+ (constVector<T>&\_vector) const;

constVector<T>operator\* (constT&scalar) const;

constVector<T>operator- (constVector<T>&\_vecctor) const;

constVector<T>operator/ (constT&scalar) const;

Vector<T>&operator\*= (constT&scalar);

Vector<T>&operator+= (constVector<T>&\_vector);

Vector<T>&operator/= (constT&scalar);

constVector<T>&operator++();

constVector<T>operator++(int);

constToperator\* (constVector<T>) const;

T&operator[] (size\_tindex) { return \_array[index]; }

friendstd::ostream&operator<<<T>(std::ostream&stream, constVector<T>&\_vector);

friendconstVector<T>operator/ (constT&scalar, constVector<T>&vector);

friendstd::istream&operator>><T> (std::istream&stream, constVector<T>&\_vector);

};

template<typenameT>

std::ostream&operator<< (std::ostream&stream, constVector<T>&\_vector) {

for (size\_ti = 0; i<\_vector.count; i++)

{

stream<<std::setw(SETW)<<\_vector.\_array[i];

}

returnstream;

}

template<typenameT>

std::istream&operator>> (std::istream&stream, constVector<T>&\_vector) {

for (size\_ti = 0; i<\_vector.count; i++)

{

stream>>\_vector.\_array[i];

}

returnstream;

}

template<typenameT>

constTVector<T>::operator\* (constVector<T>\_vector) const {

size\_t min = count <\_vector.count ? count : \_vector.count;

T temp = static\_cast<T>(0);

for (size\_ti = 0; i< min; i++)

{

temp += \_array[i] \* \_vector.\_array[i];

}

return temp;

}

template<typenameT>

Vector<T>::Vector<T>(constVector<T>&\_vector) {

this->count = \_vector.count;

this->\_array = newT[count];

memcpy(\_array, \_vector.\_array, (count \* sizeof(T)));

}

template<typenameT>

Vector<T>&Vector<T>::operator=(constVector<T>&\_vector) {

if (\_array != nullptr) {

delete[] \_array;

}

count = \_vector.count;

\_array = newT[count];

for (size\_ti = 0; i< count; i++)

{

\_array[i] = \_vector.\_array[i];

}

return \*this;

}

template<typenameT>

constVector<T>Vector<T>::operator+ (constVector<T>&\_vector) const {

size\_t max = count >\_vector.count ? count : \_vector.count;

size\_t min = count <\_vector.count ? count : \_vector.count;

Vector<T> temp(max);

size\_ti;

for (i = 0; i< min; i++)

{

temp.\_array[i] = \_array[i] + \_vector.\_array[i];

}

for (; i< count; i++)

{

temp.\_array[i] = \_array[i];

}

for (; i<\_vector.count; i++)

{

temp.\_array[i] = \_vector.\_array[i];

}

return temp;

}

template<typenameT>

Vector<T>&Vector<T>::operator+= (constVector<T>&\_vector) {

\*this = \*this + \_vector;

return \*this;

}

template<typenameT>

Vector<T>&Vector<T>::operator-= (constVector<T>&\_vector) {

\*this = \*this - \_vector;

return \*this;

}

template<typenameT>

constVector<T>Vector<T>::operator- (constVector<T>&\_vector) const {

size\_t max = count >\_vector.count ? count : \_vector.count;

size\_t min = count <\_vector.count ? count : \_vector.count;

Vector<T> temp(max);

for (size\_ti = 0; i< min; i++)

{

temp.\_array[i] = \_array[i] - \_vector.\_array[i];

}

for (size\_ti = 0; i< count; i++)

{

temp.\_array[i] = \_array[i];

}

for (size\_ti = 0; i<\_vector.count; i++)

{

temp.\_array[i] = \_vector.\_array[i];

}

return temp;

}

template<typenameT>

constVector<T>Vector<T>::operator\* (constT&scalar) const {

Vector<T> temp(count);

for (size\_ti = 0; i< count; i++)

{

temp[i] = \_array[i] \* scalar;

}

return temp;

}

template<typenameT>

constVector<T>operator\* (constT&scalar, constVector<T>right) {

returnright \* scalar;

}

template<typenameT>

constVector<T>Vector<T>::operator/ (constT&scalar) const {

Vector<T> temp(count);

for (size\_ti = 0; i< count; i++)

{

temp[i] = \_array[i] / scalar;

}

return temp;

}

template<typenameT>

constVector<T>operator/ (constT&scalar, constVector<T>&\_vector) {

Vector<T> temp(\_vector.count);

for (size\_ti = 0; i<\_vector.count; i++)

{

temp[i] = scalar / \_vector.\_array[i];

}

return temp;

}

template<typenameT>

Vector<T>&Vector<T>::operator\*= (constT&scalar) {

\*this = \*this \* scalar;

return \*this;

}

template<typenameT>

Vector<T>&Vector<T>::operator/= (constT&scalar) {

\*this = \*this / scalar;

return \*this;

}

template<typenameT>

constVector<T>&Vector<T>::operator++() {

for (size\_ti = 0; i< count; i++)

{

++\_array[i];

}

return \*this;

}

template<typenameT>

constVector<T>Vector<T>::operator++(int) {

Vector<T> temp(\*this);

for (size\_ti = 0; i< count; i++)

{

\_array[i]++;

}

return temp;

}

template<typenameT>

constboolVector<T>::operator==(constVector<T>&\_vector) const {

if (count != \_vector.count)

returnfalse;

for (size\_ti = 0; i< count; i++)

{

if (\_array[i] != \_vector.\_array[i])

returnfalse;

}

returntrue;

}

template<typenameT>

constboolVector<T>::operator!=(constVector<T>&\_vector) const {

return !(\*this == \_vector);

}

template<typenameT>

voidVector<T>::resize(size\_tsize) {

Vector<T> temp(\*this);

if (\_array != nullptr) {

delete[] \_array;

}

\_array = newT[size];

size\_t min = size<count ?size : count;

memcpy(\_array, temp.\_array, (min \* sizeof(T)));

count = size;

}

template<typenameT>

constboolVector<T>::operator<= (constVector<T>&\_vector) const {

return !(\*this>\_vector);

}

template<typenameT>

constboolVector<T>::operator>= (constVector<T>&\_vector) const {

return !(\*this<\_vector);

}

template<typenameT>

constboolVector<T>::operator< (constVector<T>&\_vector) const {

if (count >\_vector.count)

returnfalse;

if (count <\_vector.count)

returntrue;

for (size\_ti = 0; i< count; i++)

{

if (\_array[i] <\_vector.\_array[i])

returntrue;

elseif (\_array[i] >\_vector.\_array[i])

returnfalse;

}

returnfalse;

}

template<typenameT>

constboolVector<T>::operator> (constVector<T>&\_vector) const {

if (count >\_vector.count)

returntrue;

if (count <\_vector.count)

returnfalse;

for (size\_ti = 0; i< count; i++)

{

if (\_array[i] >\_vector.\_array[i])

returntrue;

elseif (\_array[i] <\_vector.\_array[i])

returnfalse;

}

returnfalse;

}

template<typenameT>

voidVector<T>::push\_back(constT&element) {

resize(count + 1);

getBack() = element;

}