|  |  |
| --- | --- |
| Российский университет транспорта (МИИТ)  Институт транспортной техники и систем управления  Кафедра «Управление и защита информации» | |
| Курсовая работа  по теме «Разработка структуры данных»  по дисциплине «Системы управления базами данных и основы построения защищенных баз данных» | |
|  | Выполнил:  Студент группы ТКИ-441 Мулов С.А.  Проверил:  к.т.н. доц. Васильева М.А. |
| Москва 2024 | |

**Оглавление**

[Задание на работу 3](#_Toc184494166)

[UML-диаграмма классов приложения 5](#_Toc184494168)

[Результаты работы приложения 6](#_Toc184494170)

[Заключение 10](#_Toc184494171)

[Приложение A 11](#_Toc184494172)

[main.cpp 11](#_Toc184494173)

[vector.cpp 14](#_Toc184494174)

[vector.h 16](#_Toc184494175)

[Test.cpp 17](#_Toc184494176)

# Задание на работу

1. Для заданной структуры данных разработать API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой. На языке C++ написать конструктор и деструктор класса разрабатываемой структуры для типа int. Отладить программу.
2. Разработать метод вывода структуры в строку. Отладить приложение. Все изменения выложить в GitHub. Завести новый Pull request. Провести рефакторинг кода в соответствии с Issue.
3. Разработать все необходимые для заданной структуры методы для хранения и поиска (если задано) элемента типа int. Отладить приложение. Все изменения выложить в GitHub. Завести новый Pull request. Провести рефакторинг кода в соответствии с Issue.
4. Структура данных для любого типа данных, реализующая CRUD — акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete). Отладить приложение. Все изменения выложить в GitHub. Завести новый Pull request. Провести рефакторинг кода в соответствии с Issue.
5. Переопределение оператора сдвига. Переопределить операторы сдвига для реализации удобного взаимодействия с потоком ввода/вывода. Реализовать (не в библиотеке классов) методы ввода/ вывода структуры из/в консоль и файл, используя операторы сдвига. Отладить приложение. Все изменения выложить в GitHub. Завести новый Pull request. Провести рефакторинг кода в соответствии с Issue.
6. Тесты. Разработать тесты на конструктор и деструктор. Тесты положить в отдельную папку с названием Tests. Отладить приложение. Все изменения выложить в GitHub. Завести новый Pull request. Провести рефакторинг кода в соответствии с Issue.
7. Полное тестирование приложения. Разработать тесты на все публичные методы разрабатываемой структуры данных. Отладить приложение. Все изменения выложить в GitHub. Завести новый Pull request. Провести рефакторинг кода в соответствии с Issue.
8. Отчет по лабораторной работе "Разработка структуры данных" Разработать отчет по лабораторной работе по ГОСТ НИР 2017 (http://docs.cntd.ru/document/1200157208). Отчет должен содержать:
9. Задание на работу
10. UML-диаграмму классов приложения
11. Листинг готового приложения в текстовом формате. Для текста кода использовать шрифт "Courier New" или "Consolas" размером 11 пт с однострочным интервалом. Программу перед этим отформатировать.
12. Результаты работы приложения в виде снимков экрана. Все рисунки должны быть крупными и четкими, иметь подписи в соответствии с ГОСТ НИР 2017. Цвет фона рисунка должен быть белым, цвет шрифта ‑ черным.

# UML-диаграмма классов приложения

Диаграмма классов представлена на рисунке 1.

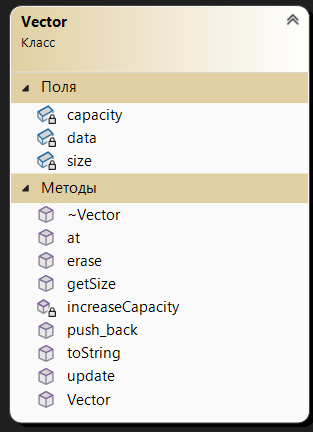


Рисунок 1 – Диаграмма классов

# Результаты работы приложения

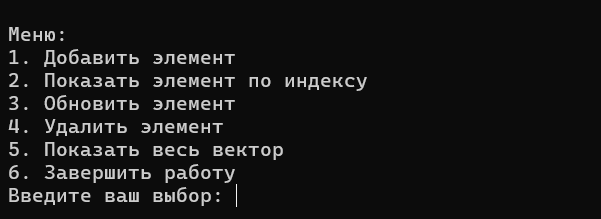


Рисунок 14 – Стартовое окно приложения

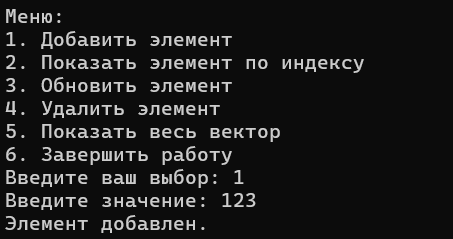


Рисунок 15 – Добавление элемента

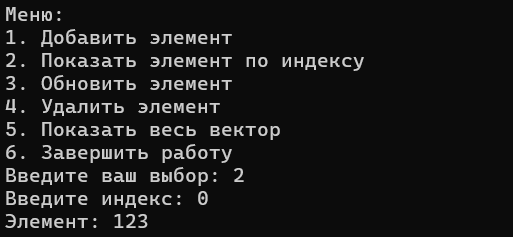


Рисунок 16 – Показ элемента по списку

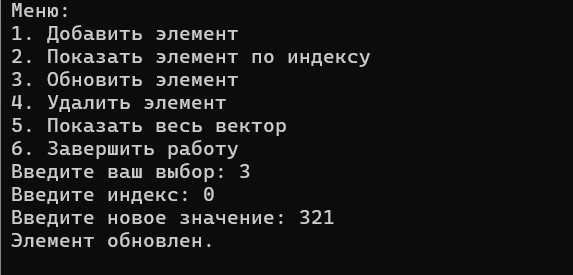


Рисунок 17 – обновление элемента

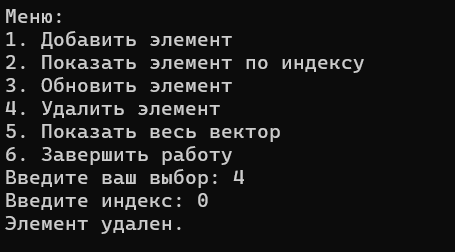


Рисунок 18 – удаление элемента

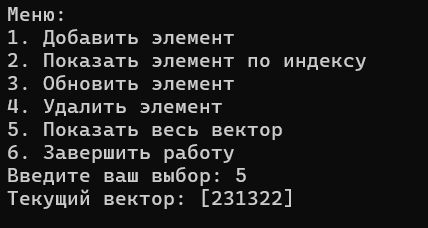
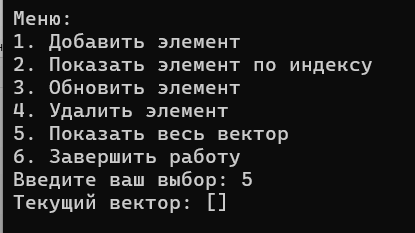


Рисунок 19 – показ всего вектора

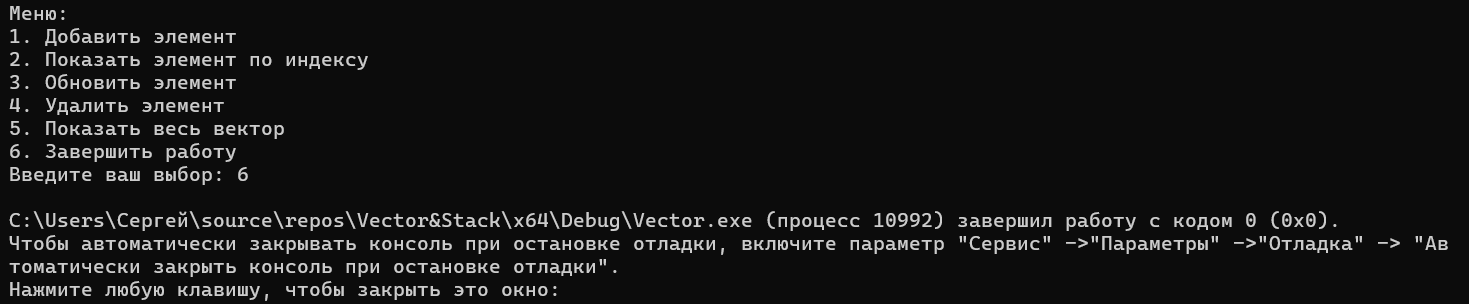


Рисунок 20 – завершение работы

Результаты тестирования методов приложения представлен на рисунке 21.

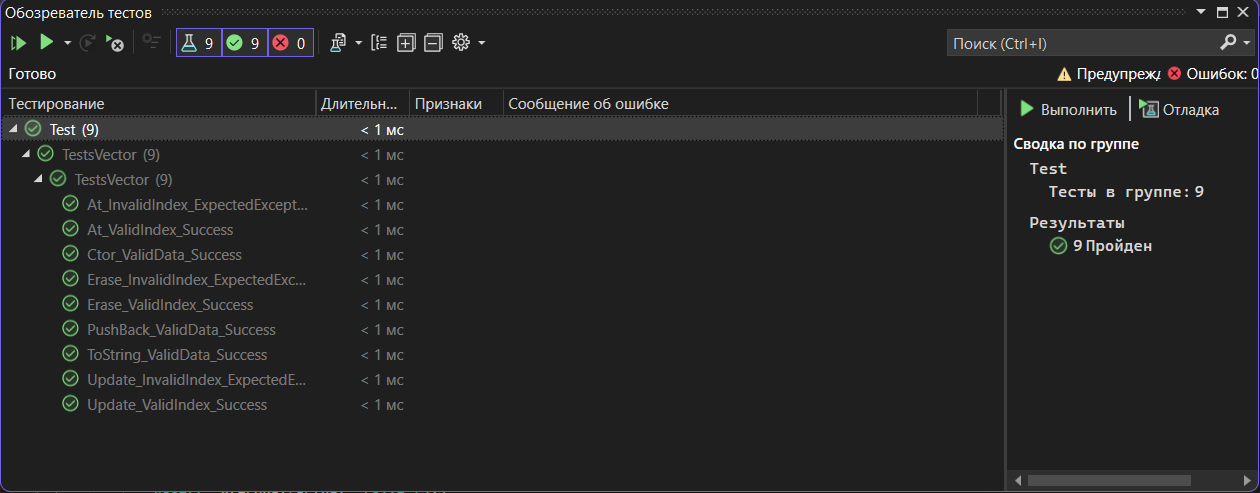


Рисунок 21 – Результаты тестирования методов приложения

# Заключение

В ходе курсовой работы было разработано API для структуры данных «Вектор». Были разработаны следующие методы:

• Конструктор вектора

• Деструктор вектора

• Метод добавления элемента в вектор

• Метод получения элемента по индексу

• Метод обновления элемента

• Метод удаления элемента

• Метод получения размера вектора

• Метод преобразования вектора в строку

Помимо этого, были разработаны и проведены тесты на методы разрабатываемой структуры данных

# Приложение A

**Листинг приложения**

main.cpp

#include "D:\Vector+Stack\Vector&Stack\Domain\Vector.cpp" // Подключение интерфейса для взаимодействия с вектором

#include <iostream> // Для стандартных потоков ввода/вывода

void handleInvalidInput(std::istream& input, std::ostream& output);

void runVectorInterface(std::istream& input, std::ostream& output);

int main() {

setlocale(LC\_ALL, "Russian");

// Запуск пользовательского интерфейса с использованием стандартных потоков

runVectorInterface(std::cin, std::cout);

return 0; // Завершение программы

}

void handleInvalidInput(std::istream& input, std::ostream& output) {

if (input.fail()) {

input.clear(); // Сбрасываем флаг ошибки

input.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Очищаем оставшиеся данные в потоке

output << "Ошибка: введено некорректное значение. Попробуйте снова.\n";

}

}

void runVectorInterface(std::istream& input, std::ostream& output) {

Vector vec; // Создаем объект вектора

bool exit = false; // Флаг завершения работы

int choice; // Переменная для хранения выбора пользователя

while (!exit) {

// Вывод меню в выходной поток

output << "\nМеню:\n";

output << "1. Добавить элемент\n";

output << "2. Показать элемент по индексу\n";

output << "3. Обновить элемент\n";

output << "4. Удалить элемент\n";

output << "5. Показать весь вектор\n";

output << "6. Завершить работу\n";

output << "Введите ваш выбор: ";

output.flush(); // Принудительная запись данных в поток

input >> choice; // Считываем выбор пользователя из входного потока

handleInvalidInput(input, output); // Проверка корректности ввода

if (input.fail()) {

continue; // Если ввод некорректный, пропускаем оставшуюся часть текущей итерации

}

switch (choice) {

case 1: { // Добавление элемента

int value;

output << "Введите значение: ";

output.flush();

input >> value;

handleInvalidInput(input, output); // Проверка корректности ввода

if (input.fail()) {

break;

}

vec.push\_back(value); // Добавляем значение в вектор

output << "Элемент добавлен.\n";

break;

}

case 2: { // Чтение элемента

size\_t index;

output << "Введите индекс: ";

output.flush();

input >> index;

handleInvalidInput(input, output); // Проверка корректности ввода

if (input.fail()) {

break;

}

try {

output << "Элемент: " << vec.at(index) << "\n"; // Выводим элемент

}

catch (const std::out\_of\_range& e) {

output << "Ошибка: " << e.what() << "\n"; // Обработка выхода за границы

}

break;

}

case 3: { // Обновление элемента

size\_t index;

int value;

output << "Введите индекс: ";

output.flush();

input >> index;

handleInvalidInput(input, output); // Проверка корректности ввода

if (input.fail()) {

break;

}

output << "Введите новое значение: ";

output.flush();

input >> value;

handleInvalidInput(input, output); // Проверка корректности ввода

if (input.fail()) {

break;

}

try {

vec.update(index, value); // Обновляем значение

output << "Элемент обновлен.\n";

}

catch (const std::out\_of\_range& e) {

output << "Ошибка: " << e.what() << "\n"; // Обработка выхода за границы

}

break;

}

case 4: { // Удаление элемента

size\_t index;

output << "Введите индекс: ";

output.flush();

input >> index;

handleInvalidInput(input, output); // Проверка корректности ввода

if (input.fail()) {

break;

}

try {

vec.erase(index); // Удаляем элемент

output << "Элемент удален.\n";

}

catch (const std::out\_of\_range& e) {

output << "Ошибка: " << e.what() << "\n"; // Обработка выхода за границы

}

break;

}

case 5: { // Показ всего вектора

output << "Текущий вектор: " << vec.toString() << "\n"; // Используем toString()

break;

}

case 6: { // Завершение работы

exit = true; // Устанавливаем флаг завершения работы

break;

}

default: {

output << "Неверный выбор. Попробуйте снова.\n";

break;

}

}

}

}

vector.cpp

#include "Vector.h" // Подключение заголовочного файла Vector

#include <sstream> // Для работы с потоками строк

// Реализация конструктора по умолчанию

Vector::Vector() : data(nullptr), size(0), capacity(0) {}

// Реализация деструктора

Vector::~Vector() {

delete[] data; // Освобождение памяти, выделенной для массива

}

// Реализация метода для увеличения вместимости

void Vector::increaseCapacity() {

capacity = (capacity == 0) ? 1 : capacity \* 2; // Увеличение вместимости в 2 раза (или установка на 1)

int\* newData = new int[capacity]; // Выделение новой памяти

for (size\_t i = 0; i < size; ++i) {

newData[i] = data[i]; // Копирование элементов

}

delete[] data; // Освобождение старой памяти

data = newData; // Переназначение указателя

}

// Реализация метода добавления элемента

void Vector::push\_back(int value) {

if (size == capacity) {

increaseCapacity(); // Увеличение вместимости, если необходимо

}

data[size++] = value; // Добавление элемента и увеличение размера

}

// Реализация метода получения элемента по индексу

int Vector::at(size\_t index) const {

if (index >= size) { // Проверка на выход за пределы массива

throw std::out\_of\_range("Index out of range"); // Генерация исключения

}

return data[index]; // Возврат элемента

}

// Реализация метода обновления элемента

void Vector::update(size\_t index, int value) {

if (index >= size) {

throw std::out\_of\_range("Index out of range");

}

data[index] = value; // Обновление элемента

}

// Реализация метода удаления элемента

void Vector::erase(size\_t index) {

if (index >= size) {

throw std::out\_of\_range("Index out of range");

}

for (size\_t i = index; i < size - 1; ++i) { // Сдвиг элементов

data[i] = data[i + 1];

}

--size; // Уменьшение размера

}

// Реализация метода получения размера

size\_t Vector::getSize() const {

return size;

}

// Реализация метода преобразования вектора в строку

std::string Vector::toString() const {

std::ostringstream oss;

oss << "[";

for (size\_t i = 0; i < size; ++i) {

oss << data[i];

if (i < size - 1) {

oss << ", ";

}

}

oss << "]";

return oss.str();

}

// Реализация перегрузки оператора вывода

std::ostream& operator<<(std::ostream& os, const Vector& vec) {

os << vec.toString(); // Вывод строки, представленной toString()

return os;

}

// Реализация перегрузки оператора ввода

std::istream& operator>>(std::istream& is, Vector& vec) {

size\_t newSize;

is >> newSize; // Чтение нового размера

vec.size = newSize;

vec.capacity = newSize;

delete[] vec.data; // Освобождение старой памяти

vec.data = new int[newSize]; // Выделение новой памяти

for (size\_t i = 0; i < newSize; ++i) {

is >> vec.data[i]; // Чтение элементов

}

return is;

}

Vector.h

#pragma VECTOR\_H

#include <iostream>

#include <stdexcept>

#include <sstream>

class Vector {

private:

int\* data; // Массив элементов

size\_t size; // Количество элементов

size\_t capacity; // Вместимость массива

void increaseCapacity(); // Увеличение вместимости

public:

Vector(); // Конструктор

~Vector(); // Деструктор

void push\_back(int value); // Добавление элемента

int at(size\_t index) const; // Доступ к элементу

void update(size\_t index, int value); // Обновление элемента

void erase(size\_t index); // Удаление элемента

size\_t getSize() const; // Получение размера

std::string toString() const; // Преобразование в строку

// Перегрузка операторов

friend std::ostream& operator<<(std::ostream& os, const Vector& vec);

friend std::istream& operator>>(std::istream& is, Vector& vec);

};

Test.cpp

#include "D:\Vector+Stack\Vector&Stack\Domain\Vector.cpp"

#include <string>

#include <stdexcept>

#include "pch.h"

#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace TestsVector

{

TEST\_CLASS(TestsVector)

{

public:

TEST\_METHOD(Ctor\_ValidData\_Success)

{

// Arrange & Act

Vector vec;

Assert::IsNotNull(&vec);

Assert::AreEqual(vec.getSize(), (size\_t)0);

}

TEST\_METHOD(PushBack\_ValidData\_Success)

{

// Arrange

Vector vec;

// Act

vec.push\_back(10);

vec.push\_back(20);

const size\_t actual = vec.getSize();

// Assert

Assert::AreEqual(actual, (size\_t)2);

}

TEST\_METHOD(At\_ValidIndex\_Success)

{

// Arrange

Vector vec;

vec.push\_back(10);

vec.push\_back(20);

// Act & Assert

Assert::AreEqual(vec.at(0), 10);

Assert::AreEqual(vec.at(1), 20);

}

TEST\_METHOD(At\_InvalidIndex\_ExpectedException)

{

// Arrange

Vector vec;

vec.push\_back(10);

// Act & Assert

Assert::ExpectException<std::out\_of\_range>(

[&vec]() { vec.at(1); });

}

TEST\_METHOD(Update\_ValidIndex\_Success)

{

// Arrange

Vector vec;

vec.push\_back(10);

vec.push\_back(20);

// Act

vec.update(0, 30);

// Assert

Assert::AreEqual(vec.at(0), 30);

}

TEST\_METHOD(Update\_InvalidIndex\_ExpectedException)

{

// Arrange

Vector vec;

vec.push\_back(10);

// Act & Assert

Assert::ExpectException<std::out\_of\_range>(

[&vec]() { vec.update(1, 20); });

}

TEST\_METHOD(Erase\_ValidIndex\_Success)

{

// Arrange

Vector vec;

vec.push\_back(10);

vec.push\_back(20);

vec.push\_back(30);

// Act

vec.erase(1);

const size\_t actual = vec.getSize();

// Assert

Assert::AreEqual(actual, (size\_t)2);

Assert::AreEqual(vec.at(1), 30);

}

TEST\_METHOD(Erase\_InvalidIndex\_ExpectedException)

{

// Arrange

Vector vec;

vec.push\_back(10);

// Act & Assert

Assert::ExpectException<std::out\_of\_range>(

[&vec]() { vec.erase(1); });

}

TEST\_METHOD(ToString\_ValidData\_Success)

{

// Arrange

Vector vec;

vec.push\_back(10);

vec.push\_back(20);

vec.push\_back(30);

// Act

std::string result = vec.toString();

}

};

}