Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе**

Дисциплина: Основы алгоритмизации и программирования

Тема: Сохранение данных в файле с использованием потоков

Выполнил работу

студент группы РИС-22-1б

Рыжков Н.С.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2023

**ТЕХНИЧЕСКАЯ СПЕЦИФИКАЦИЯ ДЛЯ ЛАБОРАТОРНОЙ РАБОТЫ №9**

Необходимо реализовать класс Vector, который будет представлять собой динамический массив целых чисел. Класс должен иметь следующие методы:

- Конструктор по умолчанию, который создает пустой массив.

- Конструктор, который создает массив заданного размера.

- Конструктор копирования.

- Оператор присваивания.

- Деструктор.

- Оператор доступа к элементам массива по индексу.

- Метод для получения размера массива.

- Метод для добавления элемента в конец массива.

- Метод для добавления элемента в начало массива.

- Оператор инкремента (prefix ++), который добавляет элемент в начало массива.

- Оператор инкремента (postfix ++), который добавляет элемент в конец массива.

**Требования к реализации**

- Класс Vector должен быть реализован на языке C++.

- Класс Vector должен быть реализован в файле vector.h.

- Класс Vector должен быть протестирован в файле main.cpp.

- Все методы класса Vector должны быть реализованы в файле vector.cpp.

- Все методы класса Vector должны быть корректно реализованы и протестированы.

**Реализация**

Реализация класса Vector должна быть разбита на два файла: vector.h и lab9\_main.cpp. В файле vector.h должно быть описание класса Vector, а в файле lab9\_main.cpp должны быть реализованы все методы класса.

**Листинг программы**

#include <stdexcept>

#include <cstring>

int A, B;

class Vector {

public:

Vector();

Vector(size\_t size);

Vector(const Vector& other);

Vector& operator=(const Vector& other);

~Vector();

int& operator[](size\_t index);

const int& operator[](size\_t index) const;

size\_t size() {

return m\_size;

};

void push\_back(int value);

void push\_front(int value);

Vector& operator++(); // prefix ++

Vector operator++(int); // postfix ++

private:

int\* m\_data;

size\_t m\_size;

size\_t m\_capacity;

void reserve(size\_t new\_capacity);

};

Vector::Vector() : m\_data(nullptr), m\_size(0), m\_capacity(0) {}

Vector::Vector(size\_t size) : m\_data(new int[size]), m\_size(size), m\_capacity(size) {}

Vector::Vector(const Vector& other) : m\_data(new int[other.m\_capacity]), m\_size(other.m\_size), m\_capacity(other.m\_capacity) {

std::memcpy(m\_data, other.m\_data, sizeof(int) \* m\_size);

}

Vector& Vector::operator=(const Vector& other) {

if (this != &other) {

if (other.m\_size <= m\_capacity) {

std::memcpy(m\_data, other.m\_data, sizeof(int) \* other.m\_size);

m\_size = other.m\_size;

} else {

Vector temp(other);

std::swap(m\_data, temp.m\_data);

std::swap(m\_size, temp.m\_size);

std::swap(m\_capacity, temp.m\_capacity);

}

}

return \*this;

}

Vector::~Vector() {

delete[] m\_data;

}

int& Vector::operator[](size\_t index) {

if (index >= m\_size) {

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

}

return m\_data[index];

}

const int& Vector::operator[](size\_t index) const {

if (index >= m\_size) {

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

}

return m\_data[index];

}

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

if (m\_size == m\_capacity) {

reserve(m\_capacity == 0 ? 1 : m\_capacity \* 2);

}

m\_data[m\_size++] = value;

}

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

if (m\_size == m\_capacity) {

reserve(m\_capacity == 0 ? 1 : m\_capacity \* 2);

}

std::memmove(m\_data + 1, m\_data, sizeof(int) \* m\_size);

m\_data[0] = value;

++m\_size;

}

Vector& Vector::operator++() { // prefix ++

push\_front(B);

return \*this;

}

Vector Vector::operator++(int) { // postfix ++

Vector temp(\*this);

push\_back(A);

return temp;

}

void Vector::reserve(size\_t new\_capacity) {

if (new\_capacity > m\_capacity) {

int\* new\_data = new int[new\_capacity];

std::memcpy(new\_data, m\_data, sizeof(int) \* m\_size);

delete[] m\_data;

m\_data = new\_data;

m\_capacity = new\_capacity;

}

}

#include <iostream>

#include <vector>

#include "vector.h"

int main() {

try {

Vector v(3);

v[0] = 1;

v[1] = 2;

v[2] = 3;

std::cin >> A ;

// добавляем элемент в конец

v++;

std::cin >> B ;

// добавляем элемент в начало

++v;

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

std::cout << v[i] << " ";

}

std::cout << "\n";

} catch (const std::exception& e) {

std::cerr << "Exception caught: " << e.what() << "\n";

}

system("pause");

return 0;

}

