ФАКУЛЬТЕТ «Информатика и системы управления»  
КАФЕДРА «Информационная безопасность»

ОТЧЕТ  
по лабораторной работе № 2  
по учебной дисциплине «Алгоритмические языки»  
на тему: «Изучение перегрузки стандартных операций в языке Си++»

Вариант 23

Выполнил:   
Студент 1 курса, гр. ИУ8-24  
Терехин Данила

**Цель работы:**

Овладение навыками разработки программ на языке C++, использующих перегрузку стандартных операций.

**Задачи работы:**

* Изучить необходимые учебные материалы, посвященные перегрузке стандартных операций в языке Си++
* Разработать программу на языке Си++ для решения заданного варианта задания
* Отладить программу
* Выполнить решение контрольного примера с помощью программы и ручной расчет контрольного примера
* Подготовить отчет по лабораторной работе

**Выполнение работы:**

Заголовочный файл **Vector.h**:

#pragma once

#include <iostream>

#include <fstream>

class Vector {

private:

    int\* data;

    int size;

public:

    // Конструкторы

    Vector();

    Vector(int\* arr, int n);

    Vector(const Vector& other); // Конструктор копирования

    Vector(Vector&& other) noexcept; // Конструктор перемещения

    // Деструктор

    ~Vector();

    // Операции

    int& operator[](int index);

    Vector& operator=(const Vector& other); // Присваивание с копированием

    Vector& operator=(Vector&& other) noexcept; // Присваивание с перемещением

    bool operator<(const Vector& other) const;

    // Ввод/вывод

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

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

    // Геттеры

    int getSize() const { return size; }

    // Сравнение с массивом

    bool isLessThan(int\* arr, int n) const;

};

**Vector.cpp**:

#include "Vector.h"

// Конструктор по умолчанию, инициализирует вектор нулевым размером и нулевым указателем на данные

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

// Конструктор, принимающий массив и его размер, инициализирует вектор данными из массива

Vector::Vector(int\* arr, int n) : size(n) {

    data = new int[size]; // Выделяет память для массива данных

    for (int i = 0; i < size; ++i) {

        data[i] = arr[i]; // Копирует данные из входного массива в вектор

    }

}

// Конструктор копирования, создает копию другого вектора

Vector::Vector(const Vector& other) : size(other.size) {

    data = new int[size]; // Выделяет память для массива данных

    for (int i = 0; i < size; ++i) {

        data[i] = other.data[i]; // Копирует данные из другого вектора

    }

}

// Конструктор перемещения, передает ресурсы из другого вектора

Vector::Vector(Vector&& other) noexcept : data(other.data), size(other.size) {

    other.data = nullptr; // Устанавливает указатель на данные другого вектора в nullptr

    other.size = 0; // Устанавливает размер другого вектора в 0

}

// Деструктор, освобождает выделенную память

Vector::~Vector() {

    delete[] data; // Освобождает память, выделенную для массива данных

}

// Оператор доступа по индексу, возвращает ссылку на элемент по указанному индексу

int& Vector::operator[](int index) {

    return data[index]; // Возвращает ссылку на элемент массива данных

}

// Оператор присваивания с копированием, копирует данные из другого вектора

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

    if (this != &other) { // Проверяет, не присваивается ли вектор самому себе

        delete[] data; // Освобождает текущие данные

        size = other.size; // Копирует размер

        data = new int[size]; // Выделяет память для нового массива данных

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

            data[i] = other.data[i]; // Копирует данные из другого вектора

        }

    }

    return \*this; // Возвращает ссылку на текущий объект

}

// Оператор присваивания с перемещением, передает ресурсы из другого вектора

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

    if (this != &other) { // Проверяет, не присваивается ли вектор самому себе

        delete[] data; // Освобождает текущие данные

        data = other.data; // Передает указатель на данные

        size = other.size; // Передает размер

        other.data = nullptr; // Устанавливает указатель на данные другого вектора в nullptr

        other.size = 0; // Устанавливает размер другого вектора в 0

    }

    return \*this; // Возвращает ссылку на текущий объект

}

// Оператор сравнения меньше, сравнивает текущий вектор с другим вектором

bool Vector::operator<(const Vector& other) const {

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

        if (data[i] >= other.data[i]) return false; // Сравнивает элементы векторов

    }

    return size < other.size; // Возвращает результат сравнения размеров векторов

}

// Функция сравнения меньше, сравнивает текущий вектор с массивом

bool Vector::isLessThan(int\* arr, int n) const {

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

        if (data[i] >= arr[i]) return false; // Сравнивает элементы вектора с элементами массива

    }

    return size < n; // Возвращает результат сравнения размеров вектора и массива

}

// Оператор ввода, считывает данные в вектор из потока ввода

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

    is >> v.size; // Считывает размер вектора

    v.data = new int[v.size]; // Выделяет память для массива данных

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

        is >> v.data[i]; // Считывает элементы вектора

    }

    return is; // Возвращает поток ввода

}

// Оператор вывода, выводит данные вектора в поток вывода

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

    os << v.size << " "; // Выводит размер вектора

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

        os << v.data[i] << " "; // Выводит элементы вектора

    }

    return os; // Возвращает поток вывода

}

**Main.cpp**

#include "Vector.h"

#include <fstream>

#include <iostream>

int main() {

    std::ifstream input("input.txt");

    std::ofstream output("output.txt");

    // Создание векторов

    Vector v1, v2;

    input >> v1 >> v2;

    output << "v1 < v2: " << (v1 < v2) << std::endl;

    int arr[] = {1, 2, 3};

    output << "v1 < arr: " << v1.isLessThan(arr, 3) << std::endl;

    // Использование конструктора копирования

    Vector v3 = v1; // Копирование v1 в v3

    output << "v3 (после копирования): " << v3 << std::endl;

    // Использование конструктора перемещения

    Vector v4(std::move(v2)); // Перемещение v2 в v4

    output << "v4 (после перемещения): " << v4 << std::endl;

    output << "v2 (после перемещения): " << v2 << std::endl; // v2 должен быть пустым

    // Использование оператора присваивания с копированием

    Vector v5;

    v5 = v1; // Копирование v1 в v5

    output << "v5 (после присваивания с копированием): " << v5 << std::endl;

    // Использование оператора присваивания с перемещением

    Vector v6;

    v6 = std::move(v3); // Перемещение v3 в v6

    output << "v6 (после присваивания с перемещением): " << v6 << std::endl;

    output << "v3 (после присваивания с перемещением): " << v3 << std::endl; // v3 должен быть пустым

    input.close();

    output.close();

    return 0;

}

Файл со входными данными – **input.txt**

3 1 2 3

4 4 5 6 7

Файл с выходными данными – **output.txt**

v1 < v2: 1

v1 < arr: 0

v3 (после копирования): 3 1 2 3

v4 (после перемещения): 4 4 5 6 7

v2 (после перемещения): 0

v5 (после присваивания с копированием): 3 1 2 3

v6 (после присваивания с перемещением): 3 1 2 3

v3 (после присваивания с перемещением): 0

**Вывод:**

В результате выполнения данной лабораторной работы я успешно овладел навыками разработки программ на языке Си++, использующих перегрузку стандартных операций. Пройдя через все этапы задачи, я углубил свои знания в области объектно-ориентированного программирования и получил практический опыт в реализации перегрузки операторов.