ФТФ, 2-ый курс, ИВТ-Б, Королёв Алексей.

**Ссылка на GitHub:** <https://github.com/ShEIH24/Lab2_Cpp>

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Тема:** Классы и объекты. Инкапсуляция. Перегрузка операторов.

**Цель:** Научиться программировать на языке С++ классы, инкапсулировать данные и методы, перегружать унарные и бинарные операторы.

**Индивидуальные задания:**

1. Написать классы Vector и Matrix для хранения и обработки одномерных и двумерных массивов, соответственно. Реализовать задание 2 лабораторной работы №1 с помощью созданных классов. Все функции оформить в виде методов классов. В коде отразить следующее:

• Выделение и освобождение динамической памяти производить в конструкторах и деструкторах классов, соответственно.

• В классе Vector перегрузить оператор индексации []. В классе Matrix добавить методы T at(int i, int j) const и setAt(int i, int j, T val), которые позволяют получить и установить значение элемента массива с индексом [i][j], T – это тип элементов массива по варианту (int или double).

• Перегрузить операторы инкремента и декремента (как префиксного, так и постфиксного). Смысл инкремента / декремента всего массива заключается в инкременте / декременте каждого элемента массива.

*(Задание №2 ЛР №1) Преобразование: 1D → 2D. Одномерный массив из 18 целых чисел необходимо отсортировать в порядке убывания и разложить по двумерной сетке 9х2 слева направо и сверху вниз. Инициализация: заполнить массив числами x[i] = i 2+1 и все элементы с четными индексами домножить на -1. Вывод на экран: на каждый элемент массива отвести 5 позиций.*

1. Написать класс Fraction для представления обыкновенных дробей (как отношения двух целых чисел x/y). Перегрузить операторы +, -, \*, / для дробей. Реализовать метод void reduce() для сокращения дроби, а также статические методы:

• int gcd(int n, int m) функция для нахождения наибольшего общего делителя чисел n и m;

• void printAsFraction(double decimal\_fraction)

• void printAsFraction(char\* decimal\_fraction) перегруженные методы вывода десятичной дроби в виде обыкновенной (например, при значении decimal\_fraction = 0.43 на экране должно вывестись 43/100, при 0.25 – 1/4 и т.д.).

Также реализовать в виде статического члена класса счетчик всех созданных на данный момент в программе экземпляров дробей. Продемонстрировать работу созданного класса. Создать несколько объектов дробей. Произвести операции сложения, вычитания, умножения и деления дробей. Вывести на экран результаты. Показать также результаты работы статических методов класса.

1. Написать собственный класс, в соответствии с вариантом. Продемонстрировать в коде инкапсуляцию данных, применение конструкторов без параметров и с параметрами для инициализации данных. Класс должен содержать метод serialize() для сохранения данных класса в файл и метод deserialize() для чтения данных класса из файла по умолчанию в текущей директории, а также перегруженные методы serialize(const std::string& filename) и deserialize(const std::string& filename) для работы с файлом с указанным в параметре именем.

*Вариант 5. Класс КОНДИЦИОНЕР. Данные: фирма, модель, вес, температура, режим, год выпуска. Создать 2 кондиционера в куче и проинициализировать их с помощью конструкторов с параметрами. Еще один кондиционер создать отдельно в стеке и установить его данные с помощью сеттеров. В главной функции проимитировать настройку кондиционеров персоналом помещений – установить каждому режим; если выбран режим охлаждения, то установить также температуру. Проделать эту процедуру три раза. Вывести информацию об использовании кондиционеров – режим, в котором на данный момент работает техника, среднее изменение температуры за весь период настройки.*

**Ход работы:**

1. Код программы:

#include <iostream>

#include <iomanip>

#include <algorithm>

// Шаблонный класс Vector для работы с одномерными массивами

template <typename T>

class Vector {

private:

T\* data; // Указатель на динамический массив

int size; // Размер массива

public:

// Конструктор: выделяет память для массива

Vector(int s) : size(s) {

data = new T[size];

}

// Деструктор: освобождает выделенную память

~Vector() {

delete[] data;

}

// Перегрузка оператора [] для доступа к элементам

T& operator[](int index) {

return data[index];

}

// Константная версия оператора []

const T& operator[](int index) const {

return data[index];

}

// Префиксный инкремент: увеличивает каждый элемент на 1

Vector& operator++() {

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

++data[i];

}

return \*this;

}

// Постфиксный инкремент

Vector operator++(int) {

Vector temp = \*this;

++(\*this);

return temp;

}

// Префиксный декремент: уменьшает каждый элемент на 1

Vector& operator--() {

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

--data[i];

}

return \*this;

}

// Постфиксный декремент

Vector operator--(int) {

Vector temp = \*this;

--(\*this);

return temp;

}

// Геттер для размера массива

int getSize() const {

return size;

}

// Геттер для доступа к внутреннему массиву

T\* getData() {

return data;

}

};

// Шаблонный класс Matrix для работы с двумерными массивами

template <typename T>

class Matrix {

private:

T\*\* data; // Указатель на указатель для создания 2D массива

int rows; // Количество строк

int cols; // Количество столбцов

public:

// Конструктор: выделяет память для 2D массива

Matrix(int r, int c) : rows(r), cols(c) {

data = new T \* [rows];

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

data[i] = new T[cols];

}

}

// Деструктор: освобождает выделенную память

~Matrix() {

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

delete[] data[i];

}

delete[] data;

}

// Метод для получения значения элемента

T at(int i, int j) const {

return data[i][j];

}

// Метод для установки значения элемента

void setAt(int i, int j, T val) {

data[i][j] = val;

}

// Префиксный инкремент: увеличивает каждый элемент на 1

Matrix& operator++() {

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

for (int j = 0; j < cols; ++j) {

++data[i][j];

}

}

return \*this;

}

// Постфиксный инкремент

Matrix operator++(int) {

Matrix temp = \*this;

++(\*this);

return temp;

}

// Префиксный декремент: уменьшает каждый элемент на 1

Matrix& operator--() {

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

for (int j = 0; j < cols; ++j) {

--data[i][j];

}

}

return \*this;

}

// Постфиксный декремент

Matrix operator--(int) {

Matrix temp = \*this;

--(\*this);

return temp;

}

// Геттеры для количества строк и столбцов

int getRows() const { return rows; }

int getCols() const { return cols; }

};

// Функция для инициализации вектора

void initializeArray(Vector<int>& vec) {

for (int i = 0; i < vec.getSize(); ++i) {

vec[i] = i \* i + 1;

if (i % 2 == 0) {

vec[i] \*= -1;

}

}

}

// Функция для вывода вектора

void printArray(const Vector<int>& vec) {

for (int i = 0; i < vec.getSize(); ++i) {

std::cout << std::setw(5) << vec[i];

if ((i + 1) % 9 == 0) std::cout << std::endl;

}

std::cout << std::endl;

}

// Функция для преобразования вектора в матрицу

void transformArray(Vector<int>& vec1D, Matrix<int>& matrix2D) {

// Сортировка вектора по убыванию

std::sort(vec1D.getData(), vec1D.getData() + vec1D.getSize(), std::greater<int>());

// Заполнение матрицы значениями из вектора

for (int i = 0; i < matrix2D.getRows(); ++i) {

for (int j = 0; j < matrix2D.getCols(); ++j) {

matrix2D.setAt(i, j, vec1D[i \* matrix2D.getCols() + j]);

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

const int size1D = 18;

// Создание и инициализация вектора

Vector<int> vec1D(size1D);

initializeArray(vec1D);

std::cout << "1D array:" << std::endl;

printArray(vec1D);

// Создание матрицы

Matrix<int> matrix2D(9, 2);

// Преобразование вектора в матрицу

transformArray(vec1D, matrix2D);

// Вывод матрицы

std::cout << "2D array:" << std::endl;

for (int i = 0; i < matrix2D.getRows(); ++i) {

for (int j = 0; j < matrix2D.getCols(); ++j) {

std::cout << std::setw(5) << matrix2D.at(i, j);

}

std::cout << std::endl;

}

return 0;

}

Результат программы:

1. Код программы:

#include <iostream>

#include <cmath>

#include <string>

#include <sstream>

#include <iomanip>

#include <limits>

class Fraction {

private:

int numerator; // Числитель дроби

int denominator; // Знаменатель дроби

static int instanceCount; // Статический счетчик созданных экземпляров

// Статический метод для нахождения наибольшего общего делителя (НОД)

static int gcd(int n, int m) {

return m == 0 ? n : gcd(m, n % m);

}

public:

// Конструктор с параметрами по умолчанию

Fraction(int num = 0, int den = 1) : numerator(num), denominator(den) {

if (denominator == 0) {

throw std::invalid\_argument("Знаменатель не может быть равен нулю");

}

reduce(); // Сокращаем дробь при создании

instanceCount++; // Увеличиваем счетчик экземпляров

}

// Деструктор

~Fraction() {

instanceCount--; // Уменьшаем счетчик экземпляров при удалении объекта

}

// Метод для сокращения дроби

void reduce() {

int common = gcd(std::abs(numerator), std::abs(denominator));

numerator /= common;

denominator /= common;

// Обеспечиваем, чтобы знак был у числителя, а знаменатель всегда положительный

if (denominator < 0) {

numerator = -numerator;

denominator = -denominator;

}

}

// Перегрузка оператора сложения

Fraction operator+(const Fraction& other) const {

return Fraction(numerator \* other.denominator + other.numerator \* denominator,

denominator \* other.denominator);

}

// Перегрузка оператора вычитания

Fraction operator-(const Fraction& other) const {

return Fraction(numerator \* other.denominator - other.numerator \* denominator,

denominator \* other.denominator);

}

// Перегрузка оператора умножения

Fraction operator\*(const Fraction& other) const {

return Fraction(numerator \* other.numerator, denominator \* other.denominator);

}

// Перегрузка оператора деления

Fraction operator/(const Fraction& other) const {

if (other.numerator == 0) {

throw std::runtime\_error("Деление на ноль");

}

return Fraction(numerator \* other.denominator, denominator \* other.numerator);

}

// Перегрузка оператора вывода

friend std::ostream& operator<<(std::ostream& os, const Fraction& f) {

os << f.numerator << "/" << f.denominator;

return os;

}

// Статический метод для вывода десятичной дроби в виде обыкновенной (версия для double)

static void printAsFraction(double decimal\_fraction) {

const long long PRECISION = 100; // 9 decimal places of precision

long long integral = std::floor(decimal\_fraction);

long long fractional = std::round((decimal\_fraction - integral) \* PRECISION);

Fraction result(integral \* PRECISION + fractional, PRECISION);

result.reduce();

std::cout << std::fixed << std::setprecision(9) << decimal\_fraction

<< " в виде обыкновенной дроби: " << result << std::endl;

}

// Исправленный метод printAsFraction для строки

static void printAsFraction(const char\* decimal\_fraction) {

double value = std::stod(decimal\_fraction);

printAsFraction(value);

}

static int getInstanceCount() {

return instanceCount;

}

};

int Fraction::instanceCount = 0;

int main() {

setlocale(LC\_ALL, "Russian");

// Создание объектов класса Fraction

Fraction f1(1, 2);

Fraction f2(3, 4);

// Вывод созданных дробей

std::cout << "f1 = " << f1 << std::endl;

std::cout << "f2 = " << f2 << std::endl;

// Демонстрация арифметических операций

std::cout << "f1 + f2 = " << (f1 + f2) << std::endl;

std::cout << "f1 - f2 = " << (f1 - f2) << std::endl;

std::cout << "f1 \* f2 = " << (f1 \* f2) << std::endl;

std::cout << "f1 / f2 = " << (f1 / f2) << std::endl;

// Вывод количества созданных экземпляров

std::cout << "Количество созданных экземпляров Fraction: " << Fraction::getInstanceCount() << std::endl;

// Демонстрация работы статических методов printAsFraction

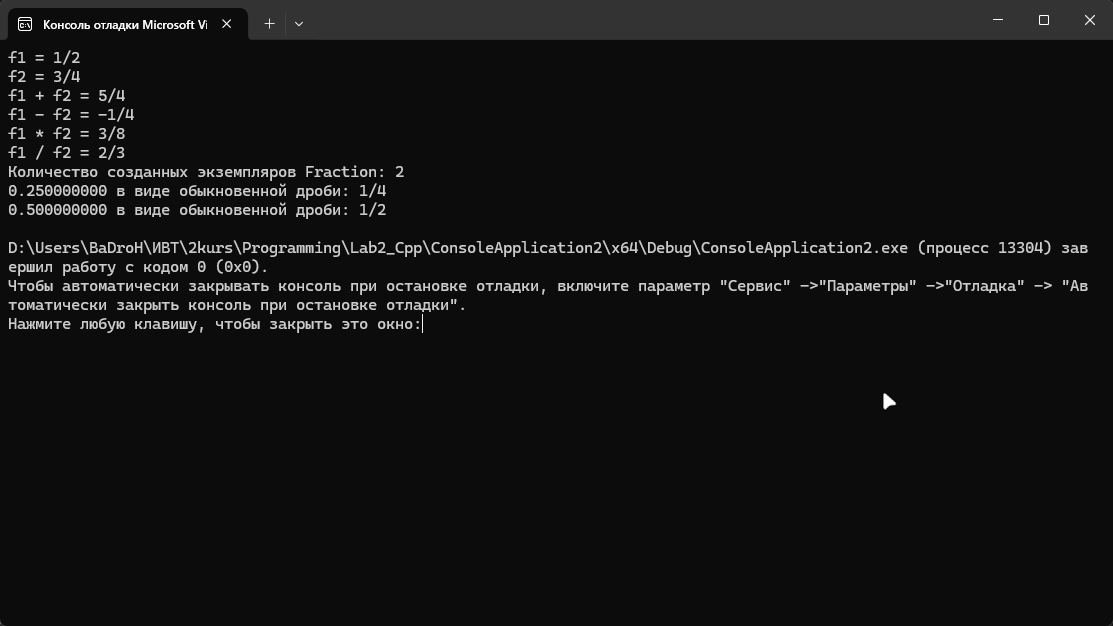
Fraction::printAsFraction(0.25);

Fraction::printAsFraction(0.50);

return 0;

}

Результат программы:



1. Код программы:
   1. Код AirConditioner.h:

#ifndef AIR\_CONDITIONER\_H

#define AIR\_CONDITIONER\_H

#include <string>

#include <fstream>

#include <iostream>

class AirConditioner {

private:

std::string brand;

std::string model;

double weight;

double temperature;

std::string mode;

int year;

double totalTempChange;

int settingsCount;

public:

// Конструктор без параметров

AirConditioner();

// Конструктор с параметрами

AirConditioner(const std::string& brand, const std::string& model, double weight, int year);

// Геттеры

std::string getBrand() const { return brand; }

std::string getModel() const { return model; }

double getWeight() const { return weight; }

double getTemperature() const { return temperature; }

std::string getMode() const { return mode; }

int getYear() const { return year; }

// Сеттеры

void setBrand(const std::string& brand) { this->brand = brand; }

void setModel(const std::string& model) { this->model = model; }

void setWeight(double weight) { this->weight = weight; }

void setYear(int year) { this->year = year; }

// Методы для настройки кондиционера

void setMode(const std::string& mode);

void setTemperature(double temperature);

// Метод для получения среднего изменения температуры

double getAverageTemperatureChange() const;

// Методы сериализации и десериализации

void serialize() const;

void deserialize();

void serialize(const std::string& filename) const;

void deserialize(const std::string& filename);

// Метод для вывода информации о кондиционере

void printInfo() const;

};

#endif // AIR\_CONDITIONER\_H

* 1. Код AirConditioner.cpp:

#include "AirConditioner.h"

#include <iostream>

#include <fstream>

#include <cmath>

// Конструктор без параметров

AirConditioner::AirConditioner() : weight(0), temperature(0), year(0), totalTempChange(0), settingsCount(0) {}

// Конструктор с параметрами

AirConditioner::AirConditioner(const std::string& brand, const std::string& model, double weight, int year)

: brand(brand), model(model), weight(weight), year(year), temperature(0), totalTempChange(0), settingsCount(0) {}

// Метод установки режима

void AirConditioner::setMode(const std::string& mode) {

this->mode = mode;

std::cout << "Режим кондиционера " << brand << " " << model << " установлен на: " << mode << std::endl;

}

// Метод установки температуры

void AirConditioner::setTemperature(double temperature) {

double tempChange = temperature - this->temperature;

this->temperature = temperature;

totalTempChange += std::abs(tempChange);

settingsCount++;

std::cout << "Температура кондиционера " << brand << " " << model << " установлена на: " << temperature << "°C" << std::endl;

}

// Метод получения среднего изменения температуры

double AirConditioner::getAverageTemperatureChange() const {

return settingsCount > 0 ? totalTempChange / settingsCount : 0;

}

// Метод сериализации (без параметров)

void AirConditioner::serialize() const {

serialize("air\_conditioner.dat");

}

// Метод десериализации (без параметров)

void AirConditioner::deserialize() {

deserialize("air\_conditioner.dat");

}

// Метод сериализации (с параметром имени файла)

void AirConditioner::serialize(const std::string& filename) const {

std::ofstream file(filename, std::ios::binary);

if (file.is\_open()) {

size\_t brandSize = brand.size();

size\_t modelSize = model.size();

size\_t modeSize = mode.size();

file.write(reinterpret\_cast<const char\*>(&brandSize), sizeof(size\_t));

file.write(brand.c\_str(), brandSize);

file.write(reinterpret\_cast<const char\*>(&modelSize), sizeof(size\_t));

file.write(model.c\_str(), modelSize);

file.write(reinterpret\_cast<const char\*>(&weight), sizeof(double));

file.write(reinterpret\_cast<const char\*>(&temperature), sizeof(double));

file.write(reinterpret\_cast<const char\*>(&modeSize), sizeof(size\_t));

file.write(mode.c\_str(), modeSize);

file.write(reinterpret\_cast<const char\*>(&year), sizeof(int));

file.write(reinterpret\_cast<const char\*>(&totalTempChange), sizeof(double));

file.write(reinterpret\_cast<const char\*>(&settingsCount), sizeof(int));

std::cout << "Данные кондиционера сохранены в файл: " << filename << std::endl;

}

else {

std::cerr << "Не удалось открыть файл для записи: " << filename << std::endl;

}

}

// Метод десериализации (с параметром имени файла)

void AirConditioner::deserialize(const std::string& filename) {

std::ifstream file(filename, std::ios::binary);

if (file.is\_open()) {

size\_t brandSize, modelSize, modeSize;

file.read(reinterpret\_cast<char\*>(&brandSize), sizeof(size\_t));

brand.resize(brandSize);

file.read(&brand[0], brandSize);

file.read(reinterpret\_cast<char\*>(&modelSize), sizeof(size\_t));

model.resize(modelSize);

file.read(&model[0], modelSize);

file.read(reinterpret\_cast<char\*>(&weight), sizeof(double));

file.read(reinterpret\_cast<char\*>(&temperature), sizeof(double));

file.read(reinterpret\_cast<char\*>(&modeSize), sizeof(size\_t));

mode.resize(modeSize);

file.read(&mode[0], modeSize);

file.read(reinterpret\_cast<char\*>(&year), sizeof(int));

file.read(reinterpret\_cast<char\*>(&totalTempChange), sizeof(double));

file.read(reinterpret\_cast<char\*>(&settingsCount), sizeof(int));

std::cout << "Данные кондиционера загружены из файла: " << filename << std::endl;

}

else {

std::cerr << "Не удалось открыть файл для чтения: " << filename << std::endl;

}

}

// Метод вывода информации о кондиционере

void AirConditioner::printInfo() const {

std::cout << "Информация о кондиционере:" << std::endl;

std::cout << "Фирма: " << brand << std::endl;

std::cout << "Модель: " << model << std::endl;

std::cout << "Вес: " << weight << " кг" << std::endl;

std::cout << "Текущая температура: " << temperature << "°C" << std::endl;

std::cout << "Режим: " << mode << std::endl;

std::cout << "Год выпуска: " << year << std::endl;

std::cout << "Среднее изменение температуры: " << getAverageTemperatureChange() << "°C" << std::endl;

}

* 1. Код основной программы:

#include "AirConditioner.h"

#include <iostream>

#include <memory>

int main() {

setlocale(LC\_ALL, "Russian");

// Создание двух кондиционеров в куче

std::unique\_ptr<AirConditioner> ac1 = std::make\_unique<AirConditioner>("Samsung", "WindFree", 12.5, 2023);

std::unique\_ptr<AirConditioner> ac2 = std::make\_unique<AirConditioner>("LG", "Artcool", 10.8, 2022);

// Создание кондиционера в стеке

AirConditioner ac3;

ac3.setBrand("Daikin");

ac3.setModel("Perfera");

ac3.setWeight(9.5);

ac3.setYear(2024);

// Имитация настройки кондиционеров персоналом (3 раза)

for (int i = 0; i < 3; ++i) {

std::cout << "\nНастройка кондиционеров (итерация " << i + 1 << "):\n";

// Настройка первого кондиционера

ac1->setMode("Охлаждение");

ac1->setTemperature(22.0 - i); // Каждый раз немного холоднее

// Настройка второго кондиционера

ac2->setMode("Вентиляция");

// Настройка третьего кондиционера

if (i % 2 == 0) {

ac3.setMode("Охлаждение");

ac3.setTemperature(24.0 - i \* 0.5);

}

else {

ac3.setMode("Осушение");

}

}

// Вывод информации об использовании кондиционеров

std::cout << "\nИтоговая информация об использовании кондиционеров:\n";

ac1->printInfo();

std::cout << std::endl;

ac2->printInfo();

std::cout << std::endl;

ac3.printInfo();

// Демонстрация сериализации и десериализации

ac1->serialize("ac1.dat");

AirConditioner loadedAC;

loadedAC.deserialize("ac1.dat");

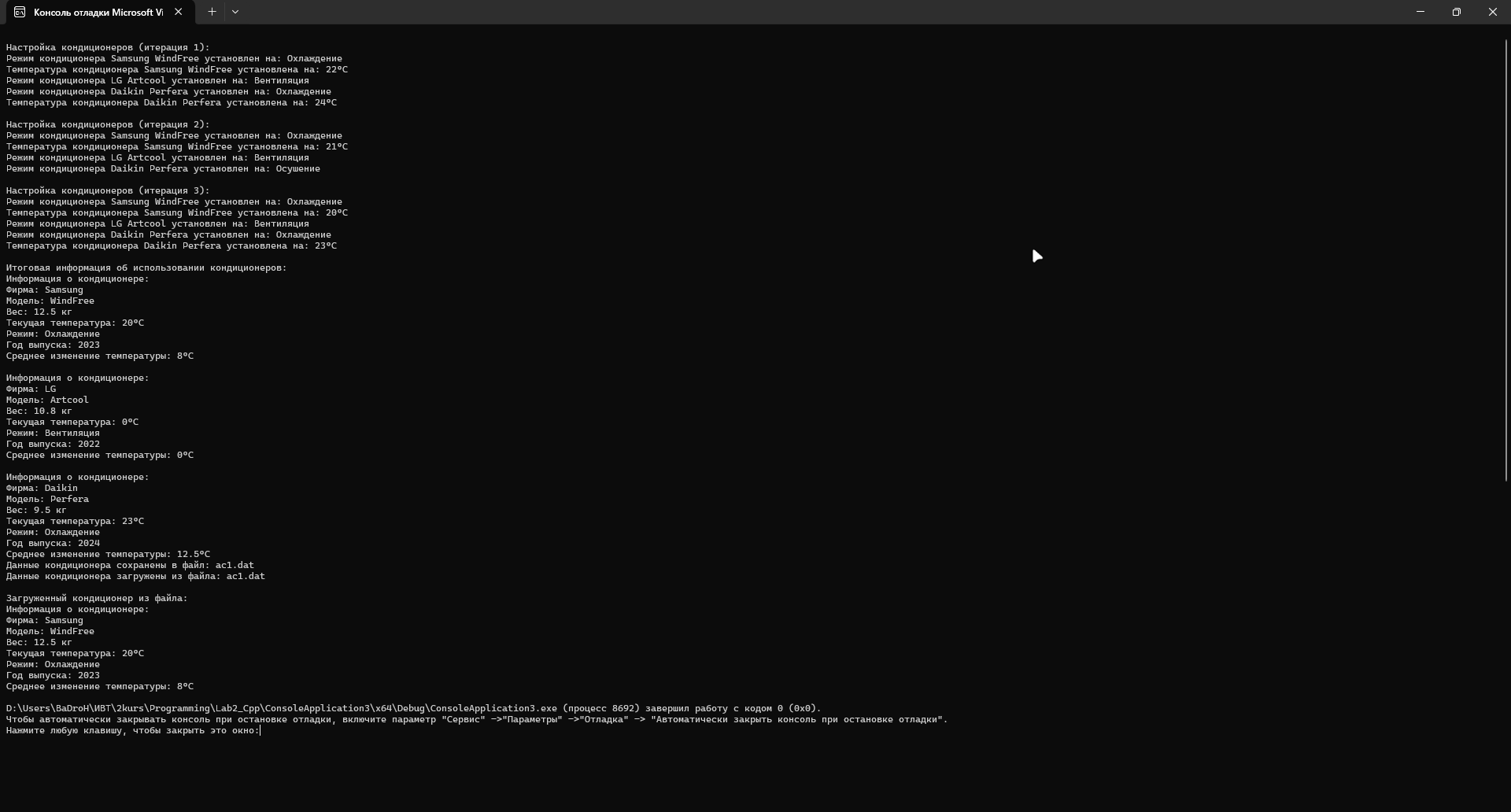
std::cout << "\nЗагруженный кондиционер из файла:\n";

loadedAC.printInfo();

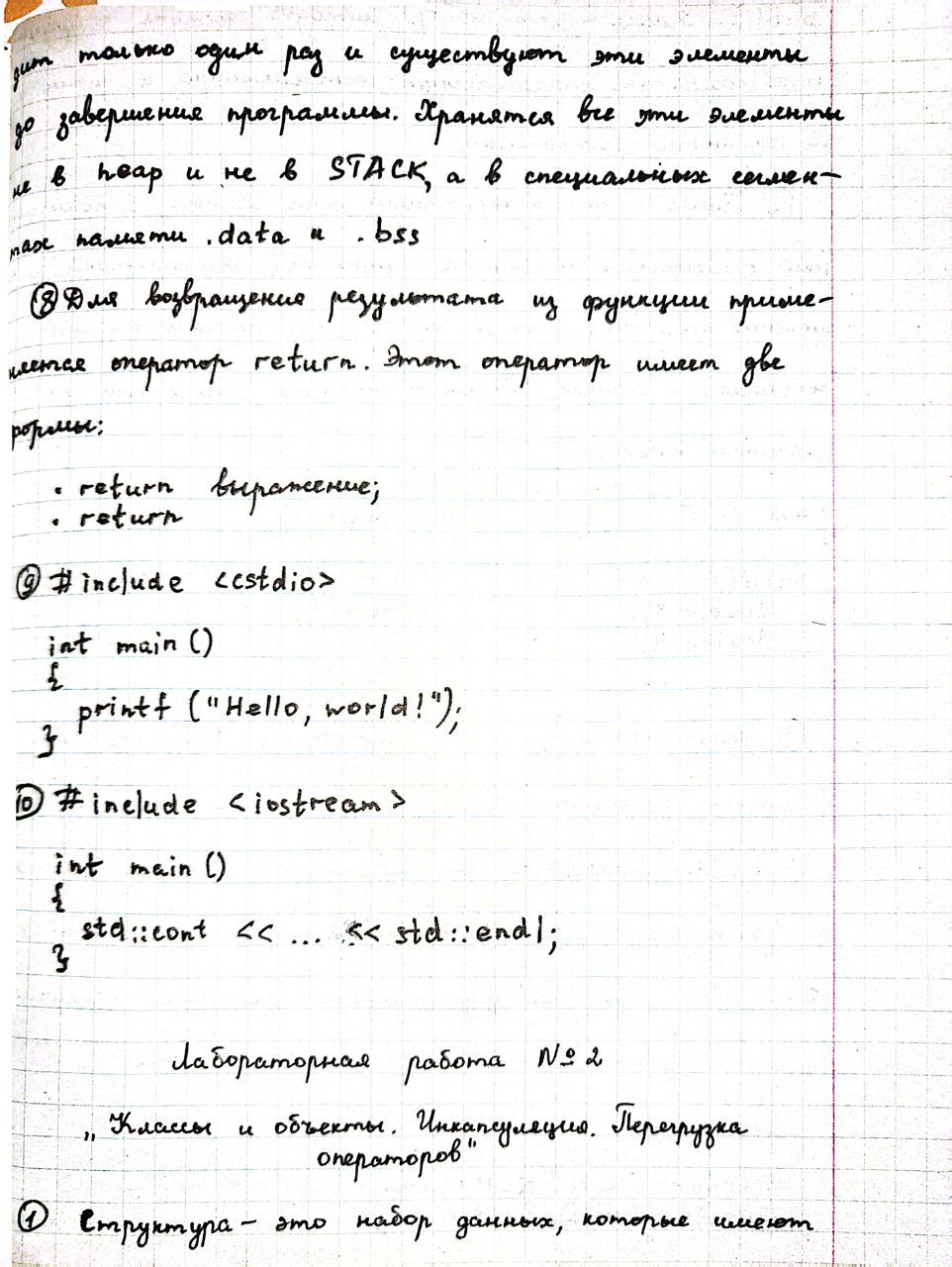
return 0;

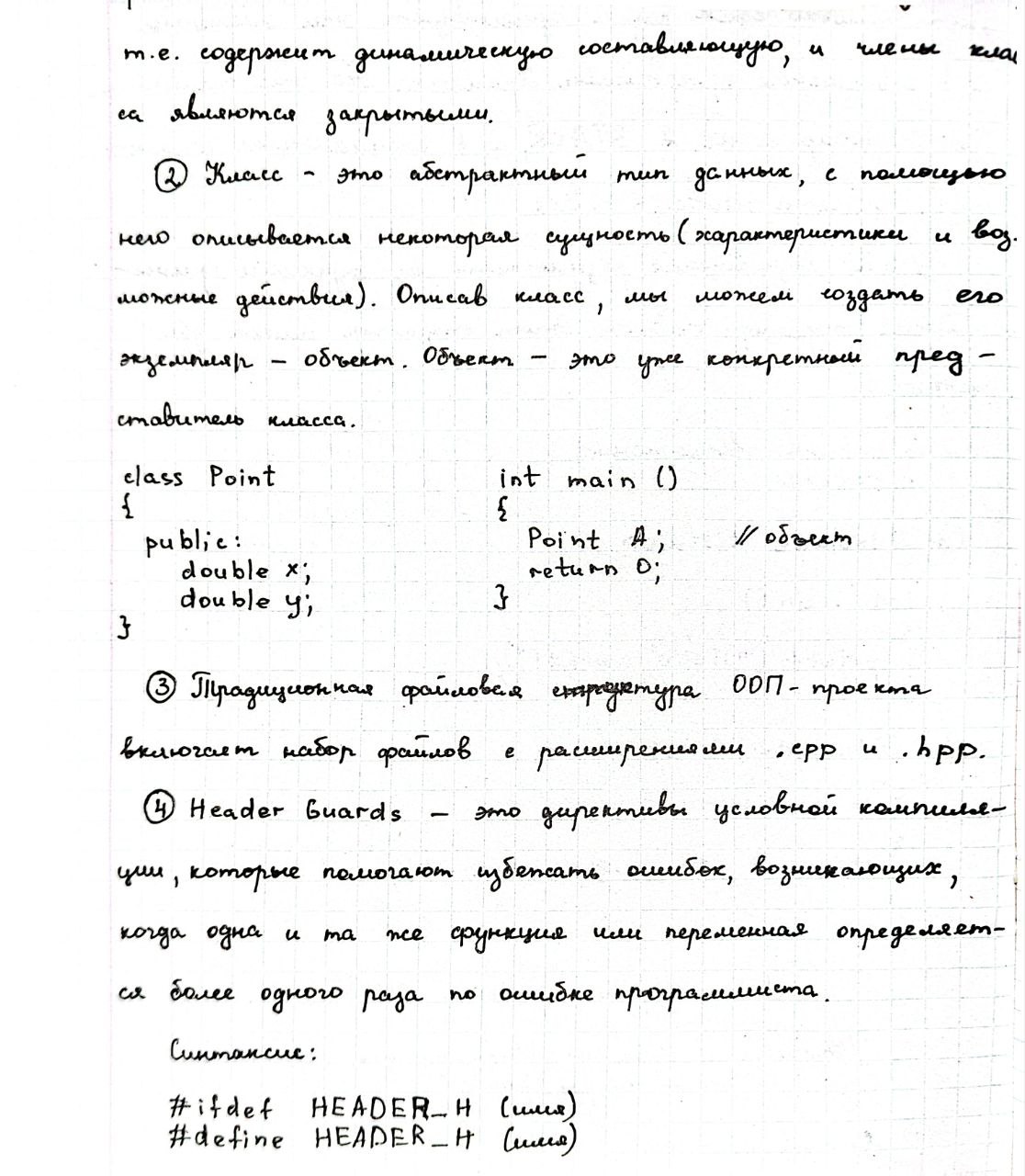
}

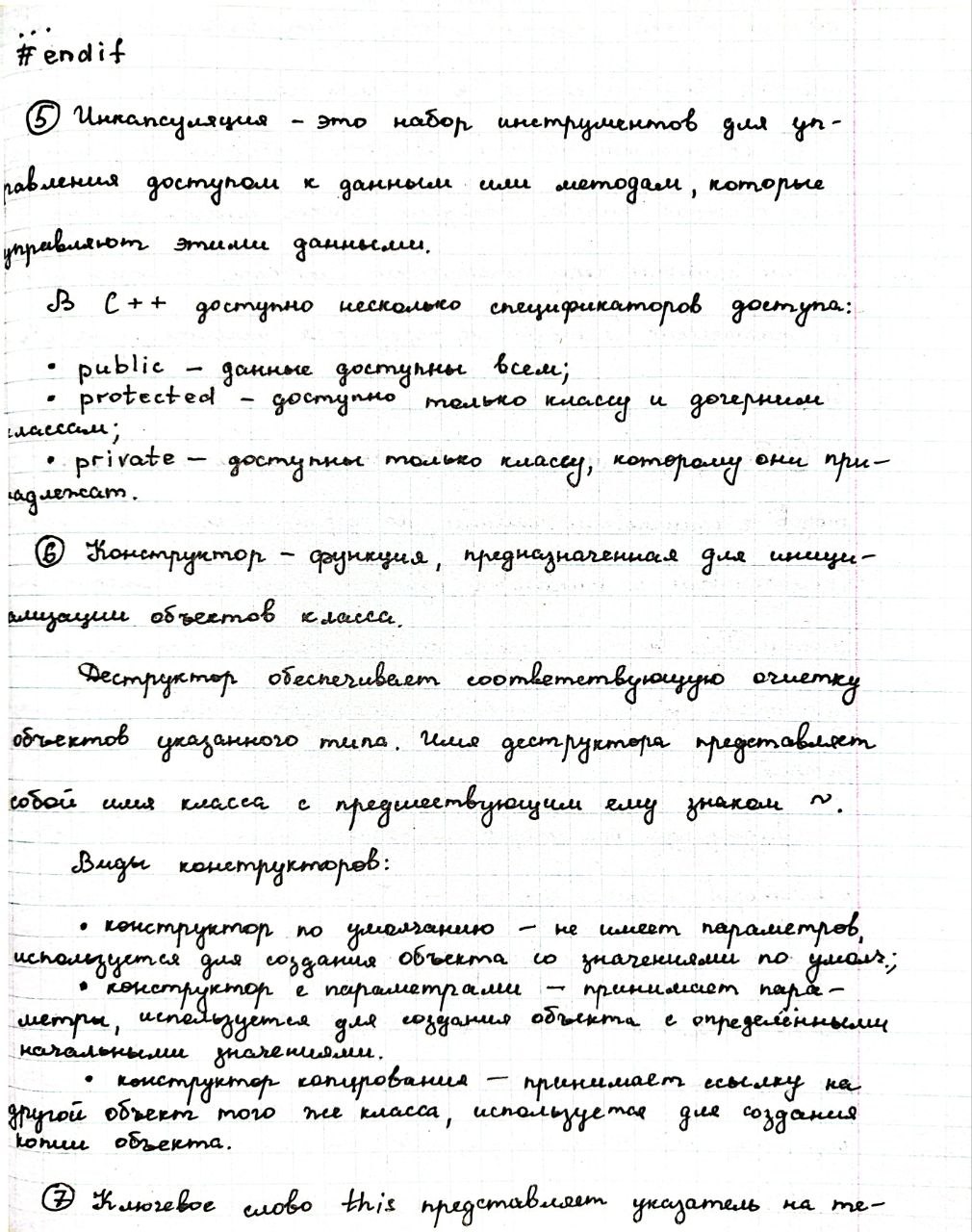
Результат программы:

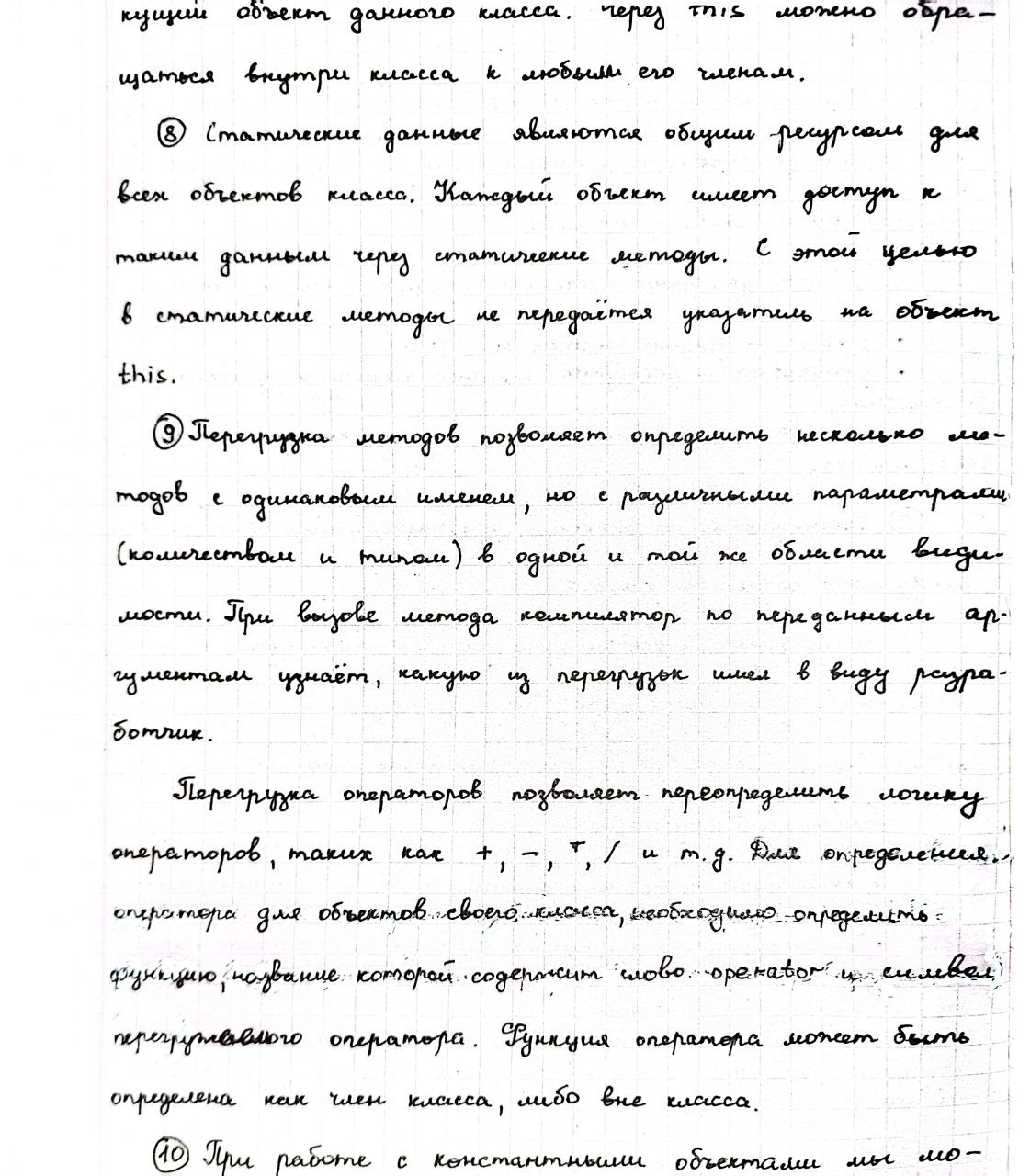


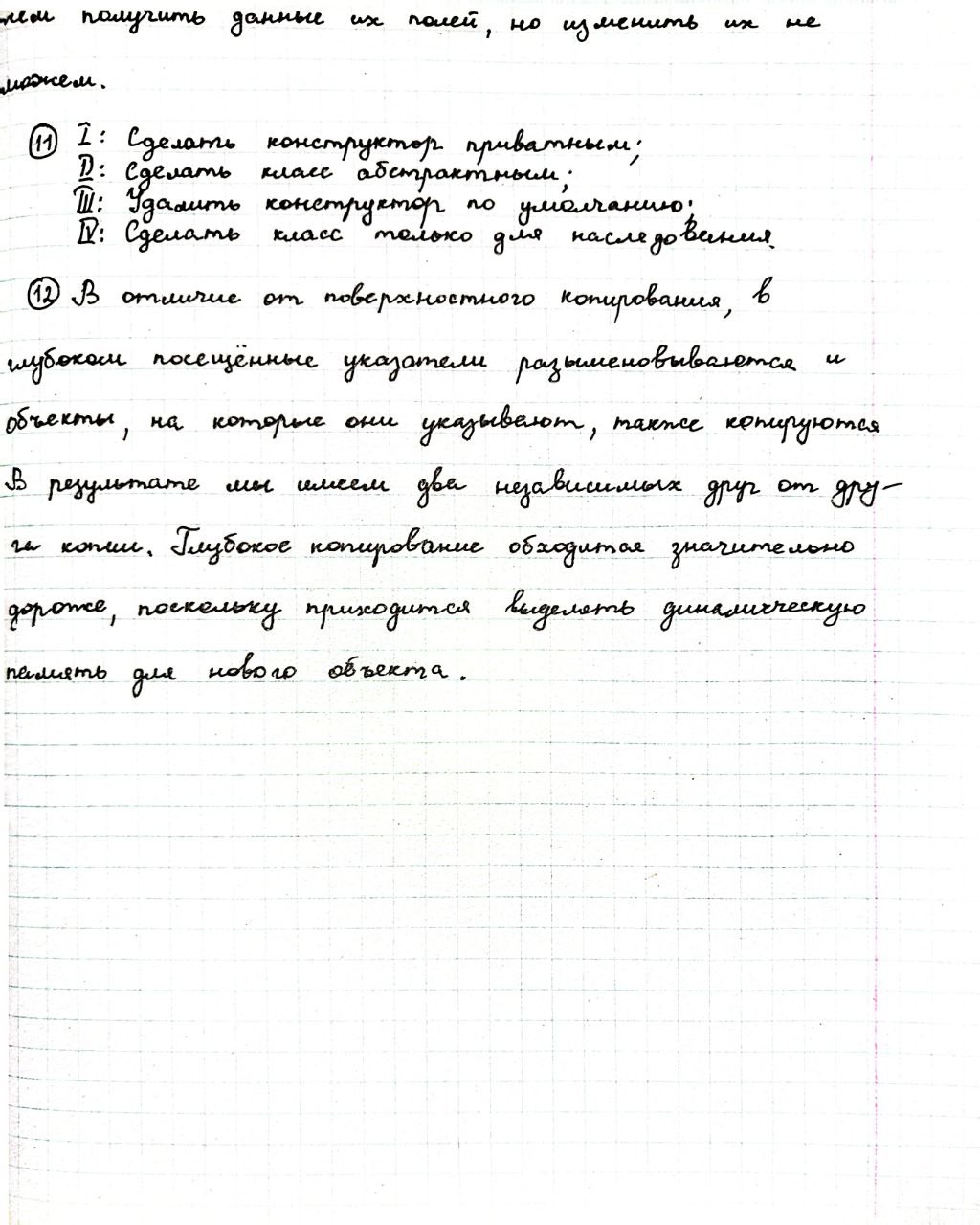
Контрольные вопросы:











Вывод: научился программировать на языке С++ классы, инкапсулировать данные и методы, перегружать унарные и бинарные операторы.