**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение высшего

образования «Кемеровский государственный университет»

институт цифры

кафедра цифровых технологий

**ОТЧЕТ**

о выполнении

Лабораторной работы №2 **«ООП на С++: инкапсуляция»**

по курсу «Языки программирования»

студента 1 курса

Коник Ильи Николаевича

(ФИО полностью)

направление подготовки 02.03.03 Математическое обеспечение и

администрирование информационных систем

|  |
| --- |
| Преподаватель: |
| Дуванов И. О. |
|  |
|  |

Кемерово 2025

# Тема: ООП на С++: инкапсуляция *(9 баллов)*

## Задание

1. В соответствии с вариантами задания и приведённой ниже спецификацией реализуйте программу на языке C++.
2. Разработайте тестовые задания, и протестируйте программу одним из методов тестирования.

## Требования к оформлению программ:

1. **Содержание**. Программа должна делать только то, что предусмотрено заданием. Не нужно выполнять лишнюю работу.
2. **Спецификация.** В преамбуле программы в комментариях указывать сведения:

* Кто выполнил.
* Что делает программа (кратко).
* Что на входе, имена входных файлов указываются.
* Что на выходе (что является результатов работы программы).

1. **Ввод и вывод**

* Приглашение пользователю. (Например, сколько чисел, какого типа и через какой разделитель нужно вводить).
* Контрольный вывод (все введенные данные выводить на экран, и только после этого выполнять необходимые вычисления.)
* «Защита от дурака». Проверять вводимые данные на корректность. Например, если необходимо считать количество чего – то, то эта величина не может быть отрицательной и т.д.

1. **Структура кода.** Набираемый код должен быть хорошо структурированным.

Использовать:

* Отступы.
* Комментарии – поясняют решение программы.
* Осмысленные названия переменных.
* Пояснения о назначении переменных в комментариях (кроме счетчиков).

1. **Декомпозиция кода**

* Функциональная
* Объектно-ориентированная

1. **Многофайловые проекты**

* Классы определять в отдельном h-файле, а все методы классов – в соответствующем cpp - файле. Созданный заголовочный файл подключать к проекту.
* Методы класса реализовывать не в определении класса.
* cpp-файл и соответствующий ему h-файл называть одинаково. В качестве названия выбирать имя того класса, который определен в соответствующем модуле.

## Работа в классе:

* Напишите класс для определения времени Time. Предусмотрите набор из трех закрытых полей: часы (hours), минуты (minutes), секунды (seconds). Реализуйте методы изменения и возврата значений закрытых полей, два конструктора – пустой и инициализирующий, деструктор, метод вывода времени в формате «чч:мм:сс» Print, метод нормализации Normalize (количество секунд и минут не должно быть больше 60, а количество часов больше 24), метод добавления заданного количества секунд ко времени AddSeconds. Протестируйте работу всех методов.
* Рассмотрите другую реализацию класса из прикрепленных файлов кода lab2. Проанализируйте работу программы.

## Задания для самостоятельного выполнения:

Напишите класс Bell, который при вызове метода sound печатает попеременно слова ding и dong, начиная c ding.

Bell.cpp

#include "Bell.h"

#include <iostream>

Bell::Bell() : isDing(true) {}

void Bell::sound() {

    if (isDing) {

        std::cout << "ding" << std::endl;

    } else {

        std::cout << "dong" << std::endl;

    }

    isDing = !isDing;

}

Bell.h

#ifndef BELL\_H

#define BELL\_H

class Bell {

private:

    bool isDing;  // Флаг для отслеживания состояния

public:

    Bell();

    void sound();

};

#endif // BELL\_H

main.cpp

#include "Bell.h"

int main() {

    Bell bell;

    bell.sound();

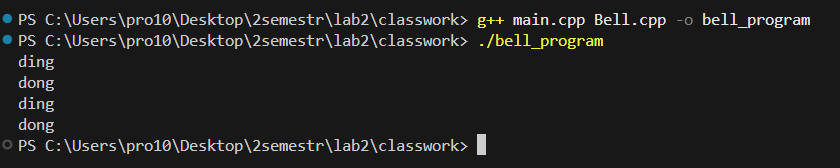
    bell.sound();

    bell.sound();

    bell.sound();

    return 0;

}



Напишите класс OddEvenSeparator, в который можно добавлять числа, получая потом отдельно чётные и нечётные. Числа добавляются в объект с помощью метода add\_number. Методы even и odd должны печатать на экране списки чётных и нечётных чисел соответственно. Числа должны идти в том же порядке, что и при добавлении в объект.

main.cpp

#include "OddEvenSeparator.h"

int main() {

    OddEvenSeparator separator;

    separator.add\_number(1);

    separator.add\_number(2);

    separator.add\_number(3);

    separator.add\_number(4);

    separator.add\_number(5);

    separator.even(); // 2 4

    separator.odd(); // 1 3 5

    return 0;

}

OddEvenSeparator.h

#ifndef ODDEVENSEPARATOR\_H

#define ODDEVENSEPARATOR\_H

#include <vector>

class OddEvenSeparator {

private:

    std::vector<int> numbers;

    std::vector<int> evens;

    std::vector<int> odds;

public:

    void add\_number(int number); // Метод для добавления числа

    void even();                // Метод для вывода чётных чисел

    void odd();                 // Метод для вывода нечётных чисел

};

#endif // ODDEVENSEPARATOR\_H

OddEvenSeparator.cpp

#include "OddEvenSeparator.h"

#include <iostream>

// Метод для добавления числа

void OddEvenSeparator::add\_number(int number) {

    numbers.push\_back(number); // Добавляем число в общий список

    if (number % 2 == 0) {

        evens.push\_back(number);

    } else {

        odds.push\_back(number);

    }

}

// Метод для вывода чётных чисел

void OddEvenSeparator::even() {

    std::cout << "Even numbers: ";

    for (int num : evens) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

}

// Метод для вывода нечётных чисел

void OddEvenSeparator::odd() {

    std::cout << "Odd numbers: ";

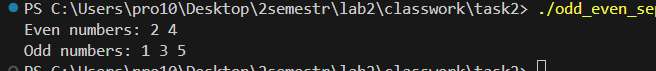
    for (int num : odds) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

}



Реализуйте класс Table, который хранит целые числа в двумерной таблице. При инициализации Table(rows, cols) экземпляру передаются число строк и столбцов в таблице. Строки и столбцы нумеруются с нуля. Ячейки таблицы инициализируются нулями. Гарантируется, что при работе с вашим классом используются только корректные индексы.

table.get\_value(row, col) — прочитать значение из ячейки со строкой row, столбцом col.

table.set\_value(row, col, value) — записать число в ячейку со строкой row, столбцом col.

table.n\_rows() — вернуть число строк в таблице.

table.n\_cols() — вернуть число столбцов в таблице.

table.print() — вывести на экран всю таблицу.

table.average() — вернуть среднее арифметическое всех значений таблицы.

main.cpp

#include "Table.h"

#include <iostream>

int main() {

    // Создаем таблицу 3x4

    Table table(3, 4);

    // Заполняем таблицу значениями

    table.set\_value(0, 0, 1);

    table.set\_value(0, 1, 2);

    table.set\_value(0, 2, 3);

    table.set\_value(0, 3, 4);

    table.set\_value(1, 0, 5);

    table.set\_value(1, 1, 6);

    table.set\_value(1, 2, 7);

    table.set\_value(1, 3, 8);

    table.set\_value(2, 0, 9);

    table.set\_value(2, 1, 10);

    table.set\_value(2, 2, 11);

    table.set\_value(2, 3, 12);

    // Выводим таблицу

    std::cout << "Table:" << std::endl;

    table.print();

    // Получаем количество строк и столбцов

    std::cout << "Rows: " << table.n\_rows() << std::endl;

    std::cout << "Columns: " << table.n\_cols() << std::endl;

    // Получаем значение из ячейки

    std::cout << "Value at (1, 2): " << table.get\_value(1, 2) << std::endl;

    // Вычисляем среднее арифметическое

    std::cout << "Average: " << table.average() << std::endl;

    return 0;

}

Table.cpp

#include "Table.h"

#include <iostream>

Table::Table(int rows, int cols) : rows(rows), cols(cols) {

    data = new int\*[rows];

    for (int i = 0; i < rows; ++i) {

        data[i] = new int[cols](); // Инициализируем нулями

    }

}

Table::~Table() {

    // Освобождаем память

    for (int i = 0; i < rows; ++i) {

        delete[] data[i];

    }

    delete[] data;

}

// Получить значение из ячейки

int Table::get\_value(int row, int col) {

    return data[row][col];

}

// Установить значение в ячейку

void Table::set\_value(int row, int col, int value) {

    data[row][col] = value;

}

// Получить количество строк

int Table::n\_rows() {

    return rows;

}

// Получить количество столбцов

int Table::n\_cols() {

    return cols;

}

// Вывести таблицу на экран

void Table::print() {

    for (int i = 0; i < rows; ++i) {

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

            std::cout << data[i][j] << " ";

        }

        std::cout << std::endl;

    }

}

// Вычислить среднее арифметическое

double Table::average() {

    double sum = 0;

    for (int i = 0; i < rows; ++i) {

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

            sum += data[i][j];

        }

    }

    return sum / (rows \* cols);

}

Table.h

#ifndef TABLE\_H

#define TABLE\_H

class Table {

private:

    int rows;

    int cols;

    int\*\* data;

public:

    Table(int rows, int cols);

    ~Table();

    int get\_value(int row, int col);       // Получить значение из ячейки

    void set\_value(int row, int col, int value); // Установить значение в ячейку

    int n\_rows();                          // Получить количество строк

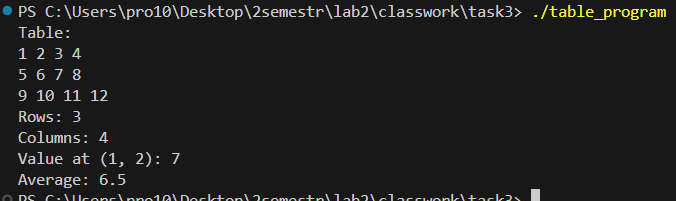
    int n\_cols();                          // Получить количество столбцов

    void print();                          // Вывести таблицу на экран

    double average();                      // Вычислить среднее арифметическое

};

#endif // TABLE\_H



Напишите класс комплексного числа Complex.

Перед выполнением задания рекомендуется изучить основные понятия комплексного числа, например, здесь:

<https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE>, <http://mathprofi.ru/kompleksnye_chisla_dlya_chainikov.html>.

В классе должно быть два поля re и im, обозначающие действительную и мнимую части комплексного числа.

В классе должно быть два конструктора: конструктор по умолчанию Complex(), который присваивает полям re и im нули, и конструктор вида Complex(double x, double y), который присваивает полям re и im значения x и y соответственно.

В классе должны быть реализованы методы возврата и изменения значений полей re и im (методы Set, Get).

В классе должны быть реализованы следующие методы:

1. double Abs() – возвращает модуль комплексного числа.
2. double Arg() – возвращает аргумент комплексного числа.
3. void Print() – выводит комплексное число на экран в стандартной форме   
   «».
4. void TrigPrint() – выводит комплексное число на экран в тригонометрической форме «».
5. void ExpPrint() – выводит комплексное число на экран в показательной форме   
   «».
6. Complex Add(Complex z) – возвращает результат сложения текущего комплексного числа с комплексным числом z в виде экземпляра класса Complex.
7. Complex Sub(Complex z) – возвращает результат вычитание из текущего комплексного числа комплексного числа z в виде экземпляра класса Complex.
8. Complex Mult(Complex z) – возвращает результат умножения текущего комплексного числа с комплексным числом z в виде экземпляра класса Complex.
9. Complex Div(Complex z) – возвращает результат деления текущего комплексного числа на комплексное число z в виде экземпляра класса Complex.

Продемонстрируйте и протестируйте работу всех методов класса.

main.cpp

#include "Complex.h"

int main() {

    // Создаем комплексные числа

    Complex z1(3, 4); // 3 + 4i

    Complex z2(1, -1); // 1 - i

    // Выводим числа

    std::cout << "z1: ";

    z1.Print();

    std::cout << "z2: ";

    z2.Print();

    // Выводим модуль и аргумент

    std::cout << "|z1| = " << z1.Abs() << std::endl;

    std::cout << "arg(z1) = " << z1.Arg() << std::endl;

    // Выводим в тригонометрической и показательной формах

    std::cout << "z1 in trigonometric form: ";

    z1.TrigPrint();

    std::cout << "z1 in exponential form: ";

    z1.ExpPrint();

    // Операции

    Complex sum = z1.Add(z2);

    Complex diff = z1.Sub(z2);

    Complex prod = z1.Mult(z2);

    Complex quot = z1.Div(z2);

    std::cout << "z1 + z2: ";

    sum.Print();

    std::cout << "z1 - z2: ";

    diff.Print();

    std::cout << "z1 \* z2: ";

    prod.Print();

    std::cout << "z1 / z2: ";

    quot.Print();

    return 0;

}

Complex.h

#ifndef COMPLEX\_H

#define COMPLEX\_H

#include <cmath>

#include <iostream>

class Complex {

private:

    double re;

    double im;

public:

    // Конструкторы

    Complex(); // По умолчанию

    Complex(double x, double y); // С параметрами

    double GetRe() const;

    double GetIm() const;

    void SetRe(double x);

    void SetIm(double y);

    // Методы

    double Abs() const; // Модуль комплексного числа

    double Arg() const; // Аргумент комплексного числа

    void Print() const; // Вывод в стандартной форме

    void TrigPrint() const; // Вывод в тригонометрической форме

    void ExpPrint() const; // Вывод в показательной форме

    Complex Add(const Complex& z) const;

    Complex Sub(const Complex& z) const;

    Complex Mult(const Complex& z) const;

    Complex Div(const Complex& z) const;

};

#endif // COMPLEX\_H

Complex.cpp

#include "Complex.h"

#include <cmath>

#include <iostream>

// Конструктор по умолчанию

Complex::Complex() : re(0), im(0) {}

// Конструктор с параметрами

Complex::Complex(double x, double y) : re(x), im(y) {}

// Получить действительную часть

double Complex::GetRe() const {

    return re;

}

// Получить мнимую часть

double Complex::GetIm() const {

    return im;

}

// Установить действительную часть

void Complex::SetRe(double x) {

    re = x;

}

// Установить мнимую часть

void Complex::SetIm(double y) {

    im = y;

}

// Модуль комплексного числа

double Complex::Abs() const {

    return std::sqrt(re \* re + im \* im);

}

// Аргумент комплексного числа

double Complex::Arg() const {

    return std::atan2(im, re);

}

// Вывод в стандартной форме

void Complex::Print() const {

    std::cout << re << " + i\*" << im << std::endl;

}

// Вывод в тригонометрической форме

void Complex::TrigPrint() const {

    double rho = Abs();

    double phi = Arg();

    std::cout << rho << "\*(cos(" << phi << ") + i\*sin(" << phi << "))" << std::endl;

}

// Вывод в показательной форме

void Complex::ExpPrint() const {

    double rho = Abs();

    double phi = Arg();

    std::cout << rho << "\*e^(i\*" << phi << ")" << std::endl;

}

// Сложение

Complex Complex::Add(const Complex& z) const {

    return Complex(re + z.re, im + z.im);

}

// Вычитание

Complex Complex::Sub(const Complex& z) const {

    return Complex(re - z.re, im - z.im);

}

// Умножение

Complex Complex::Mult(const Complex& z) const {

    return Complex(re \* z.re - im \* z.im, re \* z.im + im \* z.re);

}

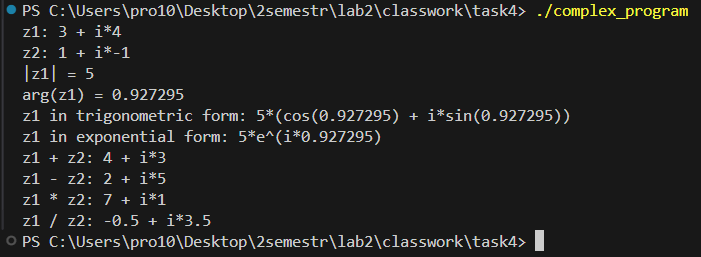
// Деление

Complex Complex::Div(const Complex& z) const {

    double denominator = z.re \* z.re + z.im \* z.im;

    return Complex((re \* z.re + im \* z.im) / denominator, (im \* z.re - re \* z.im) / denominator);

}



Вы работаете оператором на складе. Время от времени вам привозят новые коробки. Каждая коробка имеет свою грузоподъемность *wi*​ и объем *vi*​. Получая новую коробку, вы ставите на ней серийный номер, используя все целые неотрицательные числа последовательно, начиная с нуля.

Иногда вас просят выдать коробку минимальной грузоподъемности, чтобы она выдержала предмет весом *w* — или коробку минимальной вместимости, в которую можно насыпать песок объемом *v*. Вам нужно быстро определять серийный номер коробки, которая будет выдана. После выдачи коробки обратно на склад не возвращаются. Если подходящих коробок несколько, нужно выбрать ту, которая пролежала на складе меньше.

Нужно реализовать класс Stock, у которого, среди прочих, будет три метода:

1. void Add(int w, int v); — добавить коробку на склад;
2. int GetByW(int min\_w); — вернуть номер коробки грузоподъемности, хотя бы *minw*​;
3. int GetByV(int min\_v); — вернуть номер коробки объема, хотя бы *minv*​.

Если подходящей коробки нет, соответствующая функция должна вернуть −1.

Продемонстрируйте и протестируйте работу всех методов класса.

main.cpp

#include "Stock.h"

#include <iostream>

int main() {

    Stock stock;

    // Добавляем коробки

    stock.Add(10, 5); // id = 0

    stock.Add(15, 3); // id = 1

    stock.Add(20, 8); // id = 2

    stock.Add(10, 7); // id = 3

    // Тестируем GetByW

    std::cout << "GetByW(12): " << stock.GetByW(12) << std::endl; // Должен вернуть id = 1 (грузоподъемность 15)

    std::cout << "GetByW(10): " << stock.GetByW(10) << std::endl; // Должен вернуть id = 0 (грузоподъемность 10, добавлена раньше)

    // Тестируем GetByV

    std::cout << "GetByV(6): " << stock.GetByV(6) << std::endl; // Должен вернуть id = 3 (объем 7)

    std::cout << "GetByV(8): " << stock.GetByV(8) << std::endl; // Должен вернуть id = 2 (объем 8)

    // Проверяем, что коробки удаляются

    std::cout << "GetByW(10): " << stock.GetByW(10) << std::endl; // Должен вернуть -1 (подходящих коробок нет)

    return 0;

}

Stock.cpp

#include "Stock.h"

// Конструктор

Stock::Stock() : timeCounter(0) {}

// Добавить коробку

void Stock::Add(int w, int v) {

    boxes.push\_back({timeCounter, w, v, timeCounter});

    timeCounter++;

}

// Получить коробку по грузоподъемности

int Stock::GetByW(int min\_w) {

    int selectedId = -1;

    int minWeight = INT\_MAX;

    int earliestTime = INT\_MAX;

    for (const auto& box : boxes) {

        if (box.weight >= min\_w && (box.weight < minWeight || (box.weight == minWeight && box.time < earliestTime))) {

            minWeight = box.weight;

            selectedId = box.id;

            earliestTime = box.time;

        }

    }

    if (selectedId != -1) {

        // Удаляем выданную коробку

        boxes.erase(std::remove\_if(boxes.begin(), boxes.end(), [selectedId](const Box& box) {

            return box.id == selectedId;

        }), boxes.end());

    }

    return selectedId;

}

// Получить коробку по объему

int Stock::GetByV(int min\_v) {

    int selectedId = -1;

    int minVolume = INT\_MAX;

    int earliestTime = INT\_MAX;

    for (const auto& box : boxes) {

        if (box.volume >= min\_v && (box.volume < minVolume || (box.volume == minVolume && box.time < earliestTime))) {

            minVolume = box.volume;

            selectedId = box.id;

            earliestTime = box.time;

        }

    }

    if (selectedId != -1) {

        // Удаляем выданную коробку

        boxes.erase(std::remove\_if(boxes.begin(), boxes.end(), [selectedId](const Box& box) {

            return box.id == selectedId;

        }), boxes.end());

    }

    return selectedId;

}

Stock.h

#ifndef STOCK\_H

#define STOCK\_H

#include <vector>

#include <algorithm>

class Stock {

private:

    struct Box {

        int id;       // Серийный номер

        int weight;   // Грузоподъемность

        int volume;   // Объем

        int time;     // Время добавления (чем меньше, тем раньше добавлена)

    };

    std::vector<Box> boxes; // Вектор для хранения коробок

    int timeCounter;        // Счетчик времени для определения порядка добавления

public:

    Stock(); // Конструктор

    // Методы

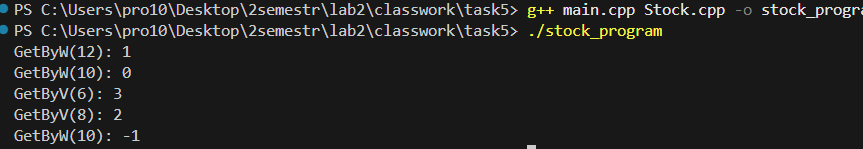
    void Add(int w, int v); // Добавить коробку

    int GetByW(int min\_w);  // Получить коробку по грузоподъемности

    int GetByV(int min\_v);  // Получить коробку по объему

};

#endif // STOCK\_H



**Критерии оценки:**

Если программа компилируется и оформлена по требованиям, то за лабораторную работу начисляются баллы.

Максимально можно получить 9 баллов:

* Задания 1,2 максимально оцениваются в 0,5 баллов
* Задание 3­– в 1 балл
* Задание 4­– в 3 балла
* Задание 5­– в 4 балла