# 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 №3 **«ООП на С++: наследование и полиморфизм»**

по курсу «Языки программирования»

студента 1 курса

Коник Ильи Николаевича

(ФИО полностью)

направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

направленность (профиль) подготовки «Интеллектуальные информационные системы и анализ данных»

Преподаватель:

Дуванов И.О.

Кемерово 2025

# Тема: ООП на С++: наследование и полиморфизм *(21 балл)*

## 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).

# 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).

# Декомпозиция кода

* + - Функциональная
    - Объектно-ориентированная

# 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:

1. Реализуйте иерархию геометрических фигур. Базовый класс Фигура (Figure). Наследники: круг (Circle), прямоугольник (Rectangle).
2. Поля всех классов должны быть закрытыми и динамическими. Для классов иерархии определите виртуальные методы вычисления площади double CalcArea() и вывода информации о фигуре void Show().
3. Базовый класс иерархии должен быть абстрактным.
4. Реализуйте алгоритм вычисления объема обобщенного цилиндра в виде полиморфной функции. Обобщенным цилиндром называется объемная фигура, в основании которой лежит произвольная плоская фигура. Объем обобщенного цилиндра вычисляется как площадь основания, умноженная на его высоту.
5. В главной программе необходимо создавать с помощью позднего связывания одну фигуру, тип которой определяет пользователь во время запуска программы. Для созданной фигуры необходимо выводить информацию, вычислять площадь и объем обобщенного цилиндра.
6. Все классы должны быть описаны в соответствующих .h и .cpp файлах.

## Задания для самостоятельного выполнения:

***Задание 1.***

Выполните задание для работы в классе, добавьте помимо классов наследников Circle и Rectangle еще два класса фигур наследников. Продемонстрируйте работу программы с учетом внесенных изменений.

Circle.cpp

//

// Created by ivan on 3/14/25.

//

#include "Circle.h"

#include <iostream>

const double M\_PI = 3.14159265358979323846;

Circle::Circle(double radius) {

    this->radius = new double(radius);

}

double Circle::calc\_area() {

    double area = M\_PI \* \*radius \* \*radius;

    return area;

}

void Circle::show() {

    std::cout << "Circle " << "\n" << "Radius :" << \*radius << "\n"

              << "Area :" << Circle::calc\_area() << "\n";

}

Circle::~Circle() {

    delete radius;

}

Circle.h

//

// Created by ivan on 3/14/25.

//

#ifndef LAB\_2\_3\_CIRCLE\_H

#define LAB\_2\_3\_CIRCLE\_H

#include "Figure.h"

class Circle : public Figure {

    double\* radius;

public:

    Circle(double radius);

    ~Circle() override;

    double calc\_area() override;

    void show() override;

};

#endif //LAB\_2\_3\_CIRCLE\_H

Figure.h

//

// Created by ivan on 3/14/25.

//

#ifndef LAB\_2\_3\_FIGURE\_H

#define LAB\_2\_3\_FIGURE\_H

class Figure {

public:

    virtual ~Figure() = default;

    virtual double calc\_area() = 0;

    virtual void show() = 0;

};

#endif //LAB\_2\_3\_FIGURE\_H

Rectangle.cpp

//

// Created by ivan on 3/14/25.

//

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(double height, double width) {

    this-> height = new double (height);

    this-> width = new double (width);

}

Rectangle::~Rectangle(){

  delete height;

  delete width;

}

double Rectangle::calc\_area() {

    return \*height \* \*width;

}

void Rectangle::show(){

    std::cout << "Rectangle\n\theight: " << \*height

    << "\n\twidth: " << \*width << "\n\tarea: " << calc\_area() << "\n";

};

Rectangle.h

//

// Created by ivan on 3/14/25.

//

#ifndef LAB\_2\_3\_RECTANGLE\_H

#define LAB\_2\_3\_RECTANGLE\_H

#include "Figure.h"

class Rectangle : public Figure {

    double\* height;

    double\* width;

public:

    Rectangle(double height, double width);

    ~Rectangle() override;

    double calc\_area() override;

    void show() override;

};

#endif //LAB\_2\_3\_RECTANGLE\_H

#include <iostream>

#include "Circle.h"

#include "Rectangle.h"

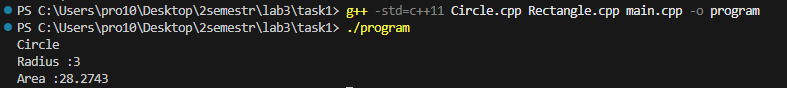
int main() {

    Circle circle(3);

    circle.show();

    return 0;

}



***Задание 2.***

Реализуйте следующую иерархию классов:

Animal – базовый класс-интерфейс с методами breathe() и eat(). Fish – наследник класса Animal с дополнительным методом swim().

Bird – наследник класса Animal с дополнительным методом lay\_eggs(). FlyingBird – наследник класса Bird с дополнительным методом fly().

Сами методы breathe, eat, swim, lay\_eggs и fly должны только выводить текст с соответствующим действием на экран.

Протестируйте работу методов для экземпляров всех классов. Создавайте объекты всех реализованных классов с помощью раннего и позднего связывания.

Animal.h

#ifndef ANIMAL\_H

#define ANIMAL\_H

class Animal {

public:

    virtual ~Animal() {} // Виртуальный деструктор

    // Чисто виртуальные методы (интерфейс)

    virtual void breathe() const = 0;

    virtual void eat() const = 0;

};

#endif // ANIMAL\_H

Bird.cpp

#include "Bird.h"

#include <iostream>

void Bird::breathe() const {

    std::cout << "Bird is breathing air" << std::endl;

}

void Bird::eat() const {

    std::cout << "Bird is eating seeds" << std::endl;

}

void Bird::lay\_eggs() const {

    std::cout << "Bird is laying eggs" << std::endl;

}

Bird.h

#ifndef BIRD\_H

#define BIRD\_H

#include "Animal.h"

class Bird : public Animal {

public:

    void breathe() const override;

    void eat() const override;

    void lay\_eggs() const;

};

#endif // BIRD\_H

Fish.cpp

#include "Fish.h"

#include <iostream>

void Fish::breathe() const {

    std::cout << "Fish is breathing underwater" << std::endl;

}

void Fish::eat() const {

    std::cout << "Fish is eating plankton" << std::endl;

}

void Fish::swim() const {

    std::cout << "Fish is swimming" << std::endl;

}

Fish.h

#ifndef FISH\_H

#define FISH\_H

#include "Animal.h"

class Fish : public Animal {

public:

    void breathe() const override;

    void eat() const override;

    void swim() const;

};

#endif // FISH\_H

FlyingBird.cpp

#include "FlyingBird.h"

#include <iostream>

void FlyingBird::fly() const {

    std::cout << "FlyingBird is flying" << std::endl;

}

FlyingBird.h

#ifndef FLYINGBIRD\_H

#define FLYINGBIRD\_H

#include "Bird.h"

class FlyingBird : public Bird {

public:

    void fly() const;

};

#endif // FLYINGBIRD\_H

Main.cpp

/\*

 \* Автор: [Ваше имя]

 \* Программа демонстрирует работу иерархии классов животных

 \* с использованием раннего и позднего связывания

 \*/

#include <iostream>

#include "Animal.h"

#include "Fish.h"

#include "Bird.h"

#include "FlyingBird.h"

void testAnimal(const Animal& animal) {

    std::cout << "\nTesting animal:" << std::endl;

    animal.breathe();

    animal.eat();

}

int main() {

    std::cout << "=== Testing with early binding ===" << std::endl;

    // Раннее связывание (статический полиморфизм)

    Fish fish;

    Bird bird;

    FlyingBird flyingBird;

    std::cout << "\nTesting Fish:" << std::endl;

    fish.breathe();

    fish.eat();

    fish.swim();

    std::cout << "\nTesting Bird:" << std::endl;

    bird.breathe();

    bird.eat();

    bird.lay\_eggs();

    std::cout << "\nTesting FlyingBird:" << std::endl;

    flyingBird.breathe();

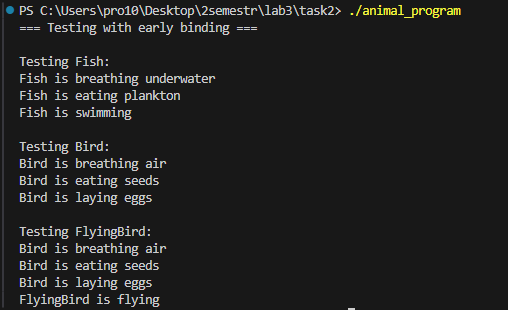
    flyingBird.eat();

    flyingBird.lay\_eggs();

    flyingBird.fly();

    return 0;

}



***Задание 3.***

Создайте базовый класс Human, который будет описывать модель человека. В нем должны храниться имя, фамилия и отчество.

Реализуйте:

* конструктор по умолчанию, в нем полям должны присваиваться значения «Неизвестно»;
* конструктор, который принимает имя, фамилия и отчество;
* метод getSurnameAndInitials, который возвращает строку вида «<Фамилия><Инициалы>», например, «Иванов И.И.»;
* метод getFullName, который возвращает строку «<Фамилия><Имя><Отчество>», например,

«Иванов Иван Иванович».

Прототип класса должен быть описан в файле «Human.h», реализация – «Human.cpp».

Создайте класс Student, который будет наследником класса Human. В нем должны храниться все оценки студента в виде вектора целых чисел.

Реализуйте:

* конструктор по умолчанию, в нем инициализируется пустой вектор;
* конструктор, который принимает имя, фамилия, отчество и вектор с оценками;
* метод addScore, который добавляет одну оценку;
* метод getAverageScore, который вычисляет и возвращает среднее арифметическое всех оценок студента;
* метод getShortStudentInfo, который возвращает строку вида

«<Фамилия><Инициалы>:<Средний балл>», например, «Иванов И.И.: 3.75»;

* метод getFullStudentInfo, который возвращает строку «<Фамилия><Имя><Отчество>:<Все оценки> – <Средний балл>», например, «Иванов Иван Иванович: 4 4 5 5 –4.5».

Прототип класса должен быть описан в файле «Student.h», реализация – «Student.cpp». Для инициализации родительских полей должен использоваться конструктор родителя.

Создайте класс Teacher, который будет наследником класса Human. В нем должны храниться:

* должность (тип - перечисление) – ассистент, старший преподаватель, доцент, профессор;
* список преподаваемых предметов в виде вектора строк.

Реализуйте:

* конструктор по умолчанию, в нем инициализируются должность «Неизвестно» и пустой вектор;
* конструктор, который принимает имя, фамилия, отчество, должность и вектор с названиями преподаваемых предметов;
* метод addSubject, который добавляет один предмет;
* метод getShortTeacherInfo, который возвращает строку вида

«<Фамилия><Инициалы><Должность>: <Количество предметов>», например, «Иванов И.И. доцент:7»;

* метод getFullTeacherInfo, который возвращает строку

«<Фамилия><Имя><Отчество><Должность>: <Все предметы через запятую>», например,

«Иванов Иван Иванович доцент: ОС, Языки программирования, Информатика».

Прототип класса должен быть описан в файле «Teacher.h», реализация – «Teacher.cpp». Для инициализации родительских полей должен использоваться конструктор родителя.

В основной программе (файл main.cpp) определите 2 массива: students (со списком студентов) и teachers(со списком преподавателей). В массивах должно быть не менее 10 записей.

Реализуйте функции для сохранения / загрузки списков студентов и преподавателей в файл / из файла. Продемонстрируйте их работу.

Выведите на экран:

1. краткую информацию обо всех студентах, у которых средний балл больше 4;
2. краткую информацию обо всех студентах, у которых средний балл меньше 3;
3. полную информацию о студенте, у которого больше всего оценок;
4. краткую информацию обо всех доцентах;
5. полную информацию о профессоре, у которого больше всего предметов;
6. полную информацию обо всех преподавателях, которые ведут информатику.

Human.cpp

#include "Human.h"

#include <sstream>

Human::Human() : firstName("Unknown"), lastName("Unknown"), patronymic("Unknown") {}

Human::Human(const std::string& firstName, const std::string& lastName, const std::string& patronymic)

    : firstName(firstName), lastName(lastName), patronymic(patronymic) {}

std::string Human::getSurnameAndInitials() const {

    std::stringstream ss;

    ss << lastName << " ";

    if (!firstName.empty()) ss << firstName[0] << ".";

    if (!patronymic.empty()) ss << patronymic[0] << ".";

    return ss.str();

}

std::string Human::getFullName() const {

    return lastName + " " + firstName + " " + patronymic;

}

Human.h

#pragma once

#include <string>

class Human {

protected:

    std::string firstName;

    std::string lastName;

    std::string patronymic;

public:

    Human();

    Human(const std::string& firstName, const std::string& lastName, const std::string& patronymic);

    std::string getSurnameAndInitials() const;

    std::string getFullName() const;

};

Student.cpp

#include "Student.h"

#include <numeric>

#include <sstream>

Student::Student() : Human(), scores() {}

Student::Student(const std::string& firstName, const std::string& lastName, const std::string& patronymic, const std::vector<int>& scores)

    : Human(firstName, lastName, patronymic), scores(scores) {}

void Student::addScore(int score) {

    scores.push\_back(score);

}

double Student::getAverageScore() const {

    if (scores.empty()) return 0.0;

    double sum = std::accumulate(scores.begin(), scores.end(), 0.0);

    return sum / scores.size();

}

std::string Student::getShortStudentInfo() const {

    std::stringstream ss;

    ss << getSurnameAndInitials() << ": " << getAverageScore();

    return ss.str();

}

std::string Student::getFullStudentInfo() const {

    std::stringstream ss;

    ss << getFullName() << ": ";

    for (int score : scores) ss << score << " ";

    ss << "- " << getAverageScore();

    return ss.str();

}

Student.h

#pragma once

#include "Human.h"

#include <vector>

class Student : public Human {

private:

    std::vector<int> scores;

public:

    Student();

    Student(const std::string& firstName, const std::string& lastName, const std::string& patronymic, const std::vector<int>& scores);

    void addScore(int score);

    double getAverageScore() const;

    std::string getShortStudentInfo() const;

    std::string getFullStudentInfo() const;

    const std::vector<int>& getScores() const { return scores; }

};

Teacher.cpp

#include "Teacher.h"

#include <sstream>

#include <map>

const std::map<Position, std::string> POSITION\_NAMES = {

    {Position::ASSISTANT, "Assistant"},

    {Position::SENIOR\_LECTURER, "Senior Lecturer"},

    {Position::DOCENT, "Docent"},

    {Position::PROFESSOR, "Professor"},

    {Position::UNKNOWN, "Unknown"}

};

Teacher::Teacher() : Human(), position(Position::UNKNOWN), subjects() {}

Teacher::Teacher(const std::string& firstName, const std::string& lastName, const std::string& patronymic, Position position, const std::vector<std::string>& subjects)

    : Human(firstName, lastName, patronymic), position(position), subjects(subjects) {}

void Teacher::addSubject(const std::string& subject) {

    subjects.push\_back(subject);

}

std::string Teacher::getShortTeacherInfo() const {

    std::stringstream ss;

    ss << getSurnameAndInitials() << " " << POSITION\_NAMES.at(position) << ": " << subjects.size() << " subjects";

    return ss.str();

}

std::string Teacher::getFullTeacherInfo() const {

    std::stringstream ss;

    ss << getFullName() << " " << POSITION\_NAMES.at(position) << ": ";

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

        if (i != 0) ss << ", ";

        ss << subjects[i];

    }

    return ss.str();

}

Teacher.h

#pragma once

#include "Human.h"

#include <vector>

#include <string>

enum class Position {

    ASSISTANT,

    SENIOR\_LECTURER,

    DOCENT,

    PROFESSOR,

    UNKNOWN

};

class Teacher : public Human {

private:

    Position position;

    std::vector<std::string> subjects;

public:

    Teacher();

    Teacher(const std::string& firstName, const std::string& lastName, const std::string& patronymic, Position position, const std::vector<std::string>& subjects);

    void addSubject(const std::string& subject);

    std::string getShortTeacherInfo() const;

    std::string getFullTeacherInfo() const;

    Position getPosition() const { return position; }

    const std::vector<std::string>& getSubjects() const { return subjects; }

};

Main.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include "Student.h"

#include "Teacher.h"

// Оператор вывода для Position

std::ostream& operator<<(std::ostream& os, const Position& pos) {

    switch(pos) {

        case Position::ASSISTANT: os << "Assistant"; break;

        case Position::SENIOR\_LECTURER: os << "Senior Lecturer"; break;

        case Position::DOCENT: os << "Docent"; break;

        case Position::PROFESSOR: os << "Professor"; break;

        default: os << "Unknown";

    }

    return os;

}

void saveStudentsToFile(const std::vector<Student>& students, const std::string& filename) {

    std::ofstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file for writing students!\n";

        return;

    }

    for (const auto& student : students) {

        file << student.getFullName() << " | Scores: ";

        for (int score : student.getScores()) {

            file << score << " ";

        }

        file << "\n";

    }

    file.close();

}

void saveTeachersToFile(const std::vector<Teacher>& teachers, const std::string& filename) {

    std::ofstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file for writing teachers!\n";

        return;

    }

    for (const auto& teacher : teachers) {

        file << teacher.getFullName() << " | Position: " << teacher.getPosition() << " | Subjects: ";

        for (const auto& subject : teacher.getSubjects()) {

            file << subject << ", ";

        }

        file << "\n";

    }

    file.close();

}

int main() {

    std::vector<Student> students = {

        Student("John", "Doe", "Smith", {4, 5, 5}),

        Student("Jane", "Smith", "Brown", {3, 3, 4}),

    };

    std::vector<Teacher> teachers = {

        Teacher("Alice", "Johnson", "Lee", Position::DOCENT, {"OOP", "Algorithms"}),

        Teacher("Bob", "Williams", "Taylor", Position::PROFESSOR, {"Informatics", "Databases"}),

    };

    saveStudentsToFile(students, "students.txt");

    saveTeachersToFile(teachers, "teachers.txt");

    std::cout << "Students with average > 4:\n";

    for (const auto& student : students) {

        if (student.getAverageScore() > 4.0) {

            std::cout << student.getShortStudentInfo() << "\n";

        }

    }

    std::cout << "\nStudents with average < 3:\n";

    for (const auto& student : students) {

        if (student.getAverageScore() < 3.0) {

            std::cout << student.getShortStudentInfo() << "\n";

        }

    }

    size\_t maxScores = 0;

    const Student\* topStudent = nullptr;

    for (const auto& student : students) {

        if (student.getScores().size() > maxScores) {

            maxScores = student.getScores().size();

            topStudent = &student;

        }

    }

    if (topStudent) {

        std::cout << "\nStudent with the most scores:\n";

        std::cout << topStudent->getFullStudentInfo() << "\n";

    }

    std::cout << "\nAll Docents:\n";

    for (const auto& teacher : teachers) {

        if (teacher.getPosition() == Position::DOCENT) {

            std::cout << teacher.getShortTeacherInfo() << "\n";

        }

    }

    size\_t maxSubjects = 0;

    const Teacher\* topProfessor = nullptr;

    for (const auto& teacher : teachers) {

        if (teacher.getPosition() == Position::PROFESSOR && teacher.getSubjects().size() > maxSubjects) {

            maxSubjects = teacher.getSubjects().size();

            topProfessor = &teacher;

        }

    }

    if (topProfessor) {

        std::cout << "\nProfessor with the most subjects:\n";

        std::cout << topProfessor->getFullTeacherInfo() << "\n";

    }

    std::cout << "\nTeachers teaching Informatics:\n";

    for (const auto& teacher : teachers) {

        for (const auto& subject : teacher.getSubjects()) {

            if (subject == "Informatics") {

                std::cout << teacher.getFullTeacherInfo() << "\n";

                break;

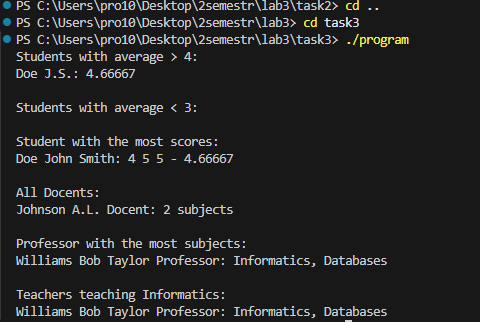
            }

        }

    }

    return 0;

}



***Задание 4.***

Реализуйте несколько классов, вычисляющих суммы следующих последовательностей натуральных чисел от 1 до N:

https://contest.yandex.ru/testsys/tex/render/MSArIDIgKyBcZG90cyArIE4=.pnghttps://contest.yandex.ru/testsys/tex/render/MSArIDJeMiArIFxkb3RzICsgTl4y.png

https://contest.yandex.ru/testsys/tex/render/MSArIDJeMyArIFxkb3RzICsgTl4z.png

Сделайте это с помощью наследования.

Базовым классом будет Summator, который реализует методы transform(i) и sum(N).

* + Метод transform(i) выполняет преобразование элемента последовательности. Для класса Summator это будет тождественное преобразование, т.е. transform(i) == i. Метод transform(i) должен быть виртуальным и будет переопределяться в производных классах.
  + Метод sum(N) реализуется в базовом классе и должен вычислять сумму значений transform(n) для всехn от 1 до N.

Производные классы должны переопределять метод transform(i):

* + SquareSummator – квадратичный сумматор, складывающий квадраты чисел, егометод transform(i) возвращает квадрат числаi2.
  + CubeSummator: – кубический сумматор, складывающий кубы чисел, егометод transform(i) возвращает куб числаi3.

Метод sum(N) переопределять не нужно.

Все классы должны быть описаны в соответствующих .h и .cpp файлах.

Подсказка: для проверки корректности реализации классов Summator, SquareSummator, CubeSummator воспользуйтесь формулами:

https://contest.yandex.ru/testsys/tex/render/MSArIDIgKyBcZG90cyArIE4gPSBcZnJhY3tOKE4gKyAxKX17Mn0=.pnghttps://contest.yandex.ru/testsys/tex/render/MSArIDJeMiArIFxkb3RzICsgTl4yID0gXGZyYWN7TihOICsgMSkoMk4gKyAxKX17Nn0=.pnghttps://contest.yandex.ru/testsys/tex/render/MSArIDJeMyArIFxkb3RzICsgTl4zID0gXGxlZnQoXGZyYWN7TihOICsgMSl9ezJ9XHJpZ2h0KV4y.png

В главной программе создавайте объекты реализованных классов с помощью раннего и позднего связывания.

CubeSummator.cpp

#include "CubeSummator.h"

int CubeSummator::transform(int i) const {

    return i \* i \* i; // Куб числа

}

CubeSummator.h

#include "Summator.h"

class CubeSummator : public Summator {

public:

    int transform(int i) const override;

};

SquareSummator.cpp

#include "SquareSummator.h"

int SquareSummator::transform(int i) const {

    return i \* i; // Квадрат числа

}

SquareSummator.h

#include "Summator.h"

class SquareSummator : public Summator {

public:

    int transform(int i) const override;

};

Summator.cpp

#include "Summator.h"

int Summator::transform(int i) const {

    return i; // Тождественное преобразование

}

int Summator::sum(int N) const {

    int total = 0;

    for (int i = 1; i <= N; ++i) {

        total += transform(i);

    }

    return total;

}

Summator.h

class Summator {

public:

    virtual ~Summator() = default;

    virtual int transform(int i) const;

    int sum(int N) const;

};

Main.cpp

#include <iostream>

#include "Summator.h"

#include "SquareSummator.h"

#include "CubeSummator.h"

void testSummator(Summator\* s, int N, const std::string& name) {

    std::cout << name << " sum from 1 to " << N << " = " << s->sum(N) << std::endl;

}

int main() {

    const int N = 5;

    // Раннее связывание

    Summator base;

    SquareSummator square;

    CubeSummator cube;

    std::cout << "Early binding results:\n";

    std::cout << "Linear sum: " << base.sum(N) << std::endl;

    std::cout << "Square sum: " << square.sum(N) << std::endl;

    std::cout << "Cube sum: " << cube.sum(N) << std::endl;

    // Позднее связывание

    Summator\* ptr1 = &base;

    Summator\* ptr2 = &square;

    Summator\* ptr3 = &cube;

    std::cout << "\nLate binding results:\n";

    testSummator(ptr1, N, "Linear");

    testSummator(ptr2, N, "Square");

    testSummator(ptr3, N, "Cube");

    // Проверка по формулам

    std::cout << "\nFormula verification:\n";

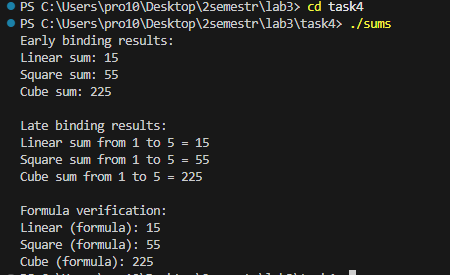
    std::cout << "Linear (formula): " << N\*(N+1)/2 << std::endl;

    std::cout << "Square (formula): " << N\*(N+1)\*(2\*N+1)/6 << std::endl;

    std::cout << "Cube (formula): " << (N\*(N+1)/2)\*(N\*(N+1)/2) << std::endl;

    return 0;

}



# Критерии оценки:

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

Максимально можно получить 21 балл:

* + Задание 1 – 2 балла
  + Задание 2 – 4 балла
  + Задание 3 - 9 баллов
  + Задание 4 – 6 баллов