Модульний контроль №1

Дисципліна: Об'єктно орієнтоване програмування

Виконав: Вівчар Вадим Вікторович, гр. АЛК-43

Факультет: Інженерія програмного забезпечення

Університет: ЗВО НМТУ

Мета роботи

Закріпити навички:

- створення ієрархій класів (наслідування);
- використання віртуальних методів (поліморфізм);
- роботи з масивами/контейнерами вказівників на базовий клас;
- застосування шаблонів функцій і класів.

Завдання 1. Ієрархія «Університет – Факультет – Кафедра». Масив вказівників. Поліморфний вивід

Постановка. Створити ієрархію класів University → Faculty → Department, сформувати контейнер вказівників на базовий клас і вивести дані про об'єкти різних типів у циклі.

Коротко про реалізацію.

- Базовий клас University містить назву університету та віртуальний метод print().
- Клас Faculty наслідує University, додає назву факультету та перевизначає print().
- Клас Department наслідує Faculty, додає назву кафедри та перевизначає print().
- Використано контейнер std::vector з вказівниками; виклик p->print() демонструє поліморфізм.
- Для коректного відображення українських символів у Windows-консолі встановлено UTF-8 (SetConsoleOutputCP/SetConsoleCP).

Ключові фрагменти коду.

```
class University {
protected:
    std::string universityName;
public:
    explicit University(std::string name) : universityName(std::move(name)) {}
    virtual void print() const { std::cout << u8"[Університет] " << universityName << "\n";
    virtual ~University() = default;
};

std::vector<University*> arr;
arr.push_back(new University(u8"ЗВО НМТУ"));
arr.push_back(new Faculty(u8"ЗВО НМТУ", u8"Інженерія програмного забезпечення"));
arr.push_back(new Department(u8"ЗВО НМТУ", u8"Інженерія програмного забезпечення",
                             u8"Кафедра інженерії програмного забезпечення"));

for (auto* p : arr) p->print();
for (auto* p : arr) delete p;
```

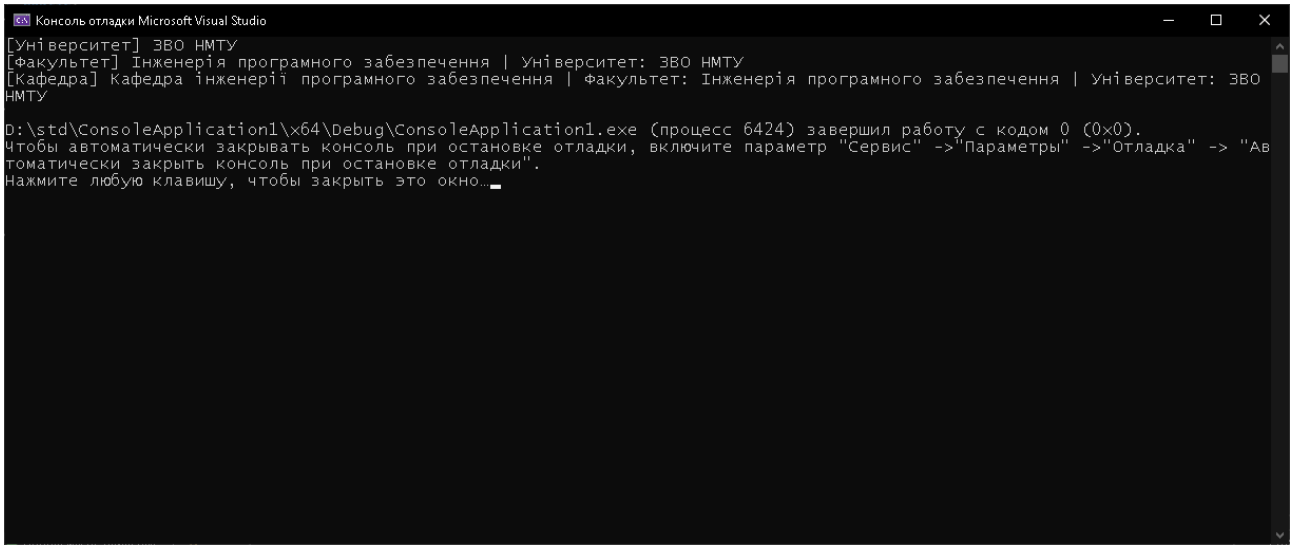

Рисунок 1 – Результат виконання програми (консоль)

Висновок до завдання 1. Реалізовано ієрархію класів та поліморфний вивід через віртуальний метод print(), використано контейнер вказівників на базовий клас.

Завдання 2. Ієрархія «Фігури обертання – Конус». Масив вказівників. Поліморфний вивід

Постановка. Створити базовий клас для фігур обертання та похідний клас Cone, сформувати контейнер вказівників і вивести характеристики об'єктів у циклі.

Коротко про реалізацію.

- Створено абстрактний базовий клас SolidOfRevolution з віртуальними методами volume(), surfaceArea(), print().
- Клас Cone реалізує обчислення об'єму та площі поверхні (l = sqrt(r^2 + h^2)).
- У Visual Studio використано власну константу PI замість M_PI для сумісності.
- Об'єкти зберігаються у контейнері std::vector (через unique_ptr); вивід здійснюється поліморфно.

Ключові фрагменти коду.

```
static constexpr double PI = 3.14159265358979323846;

class SolidOfRevolution {
public:
    virtual ~SolidOfRevolution() = default;
    virtual double volume() const = 0;
    virtual double surfaceArea() const = 0;
    virtual void print(std::ostream& os) const = 0;
};

class Cone : public SolidOfRevolution {
    double r, h;
public:
    Cone(double radius, double height) : r(radius), h(height) {}
    double slantHeight() const { return std::sqrt(r*r + h*h); }
```

```
    double volume() const override { return (PI * r*r * h) / 3.0; }
    double surfaceArea() const override { return PI * r * (r + slantHeight()); }
};
```

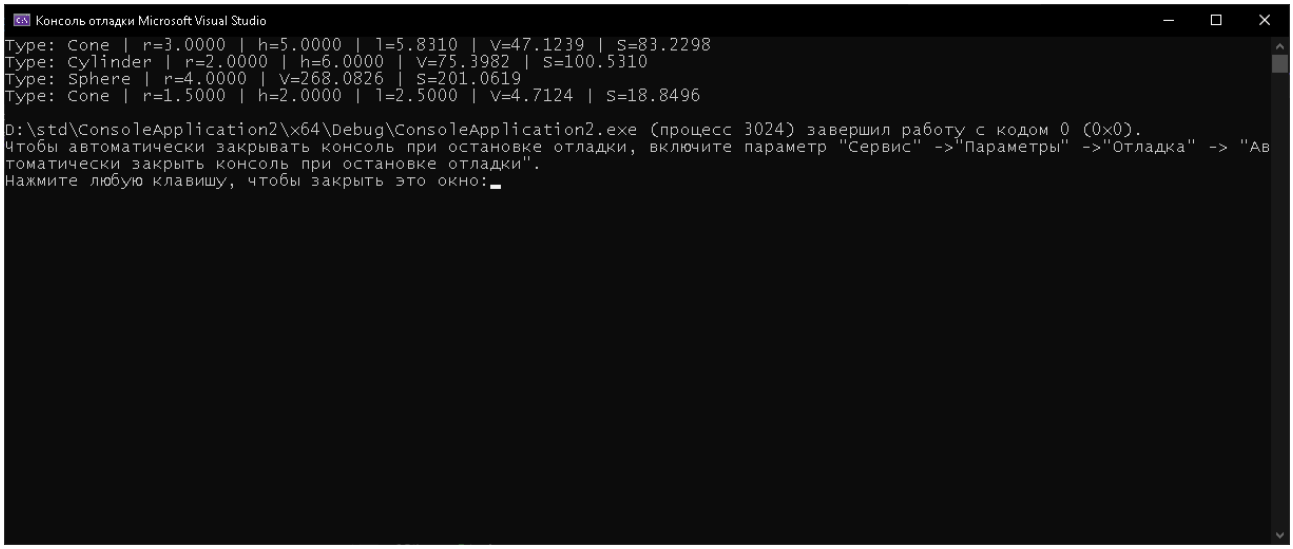

Рисунок 2 – Вивід параметрів і розрахунків для фігур обертання (консоль)

Висновок до завдання 2. Продемонстровано наслідування та поліморфізм для фігур обертання; обчислення об'єму та площі поверхні виконуються коректно.

Завдання 3. Шаблонна функція пошуку елементів масиву в заданому діапазоні

Постановка. Створити шаблонну функцію, яка знаходить елементи масиву в діапазоні [low; high] (включно) та перевірити роботу для різних типів.

Коротко про реалізацію.

- Реалізовано findInRange(arr, n, low, high), що формує вектор результатів.
- Умова належності елемента діапазону записана як low ≤ x ≤ high.
- Перевірено для типів int, double та std::string (лексикографічне порівняння).

Ключові фрагменти коду.

```
template <typename T>
std::vector<T> findInRange(const T* arr, std::size_t n, const T& low, const T& high) {
    std::vector<T> result;
    for (std::size_t i = 0; i < n; ++i) {
        if (!(arr[i] < low) && !(high < arr[i])) { // low <= arr[i] <= high
            result.push_back(arr[i]);
        }
    }
    return result;
}
```

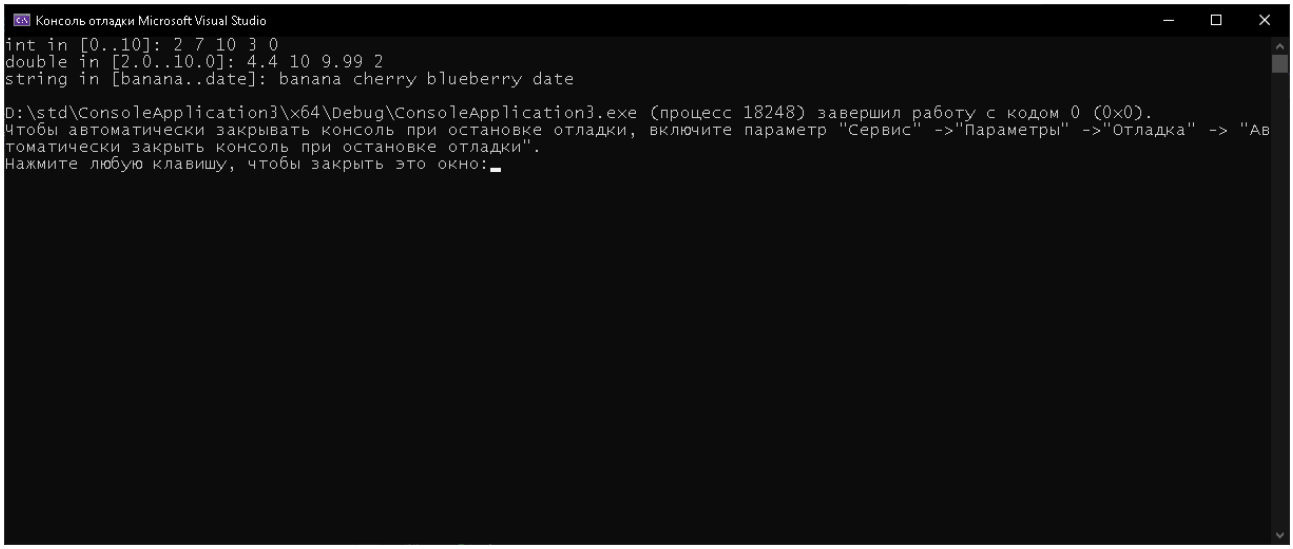

Рисунок 3 – Перевірка шаблонної функції findInRange для різних типів (консоль)

Висновок до завдання 3. Шаблонна функція коректно працює для різних типів, що підтримують операції порівняння.

Завдання 4. Шаблонний клас для зберігання пари чисел різних типів

Постановка. Створити шаблонний клас, який зберігає пару чисел різних типів, та перевірити роботу на прикладах.

Коротко про реалізацію.

- Створено клас NumberPair з полями first і second.
- Додано методи sum() та product() для обчислення суми і добутку пари.
- Типи обмежено як числові через std::is_arithmetic.

Ключові фрагменти коду.

```
template <typename T1, typename T2>
class NumberPair {
    static_assert(std::is_arithmetic_v<T1>, "T1 must be a numeric type");
    static_assert(std::is_arithmetic_v<T2>, "T2 must be a numeric type");

    T1 first;
    T2 second;

public:
    NumberPair(const T1& a, const T2& b) : first(a), second(b) {}
    auto sum() const { return first + second; }
    auto product() const { return first * second; }
};
```

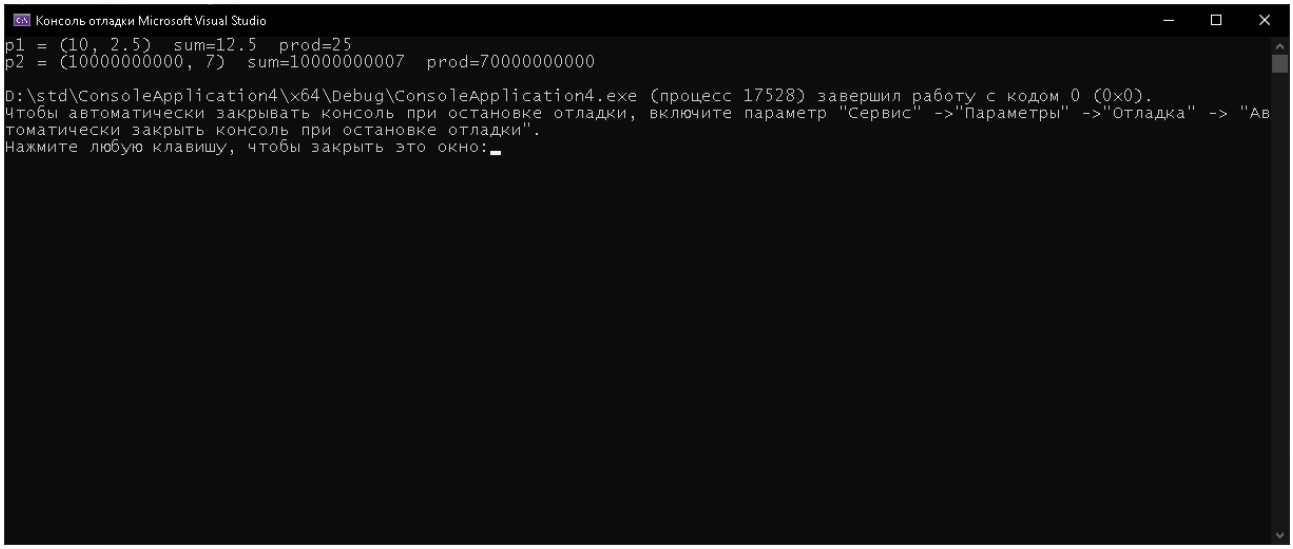

Рисунок 4 – Робота шаблонного класу NumberPair (консоль)

Висновок до завдання 4. Шаблонний клас зберігає пару різнотипних чисел і коректно виконує обчислення суми та добутку.

Загальний висновок

У роботі реалізовано наслідування, поліморфізм, роботу з контейнером вказівників, а також використання шаблонів функцій і класів. Отримані результати підтверджують коректність реалізацій, а приклади виводу демонструють роботу програм для різних типів даних.