 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №4**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Успадкування. Поліморфізм»

Варіант №15

**Виконав:**

студент гр. БС-71

Орлівський С. П.

**Перевірив:**

асистент каф. БМК

Рисін С.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2019

**🞏 Практичне заняття без зауважень**

**🞏 Практичне заняття має зауваження:**

**🞏 несвоєчасний захист**

**🞏 присутні зауваження до UML діаграми:**

**🞏 діаграма класу не відповідає коду**

**🞏 виконані не за стандартом:**

**🞏 атрибути**

**🞏 відношення**

**🞏 потужність**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 хибно задані специфікатори доступу**

**🞏 помилки у визначенні конструкторів / деструкторів**

**🞏 відсутні списки ініціалізації в конструкторах**

**🞏 константні методи**

**🞏 використано глобальні змінні**

**🞏 статичні змінні при роботі з масивами**

**🞏 оформлення коду**

**🞏 присутні зайві символи «{» та «}»**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 невірні відповіді на запитання:**

**🞏 №1 🞏 №2 🞏 №3 🞏 №4 🞏 №5**

**🞏 №6 🞏 №7 🞏 №8 🞏 №9 🞏 №10**

**🞏 маються інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Завдання:**

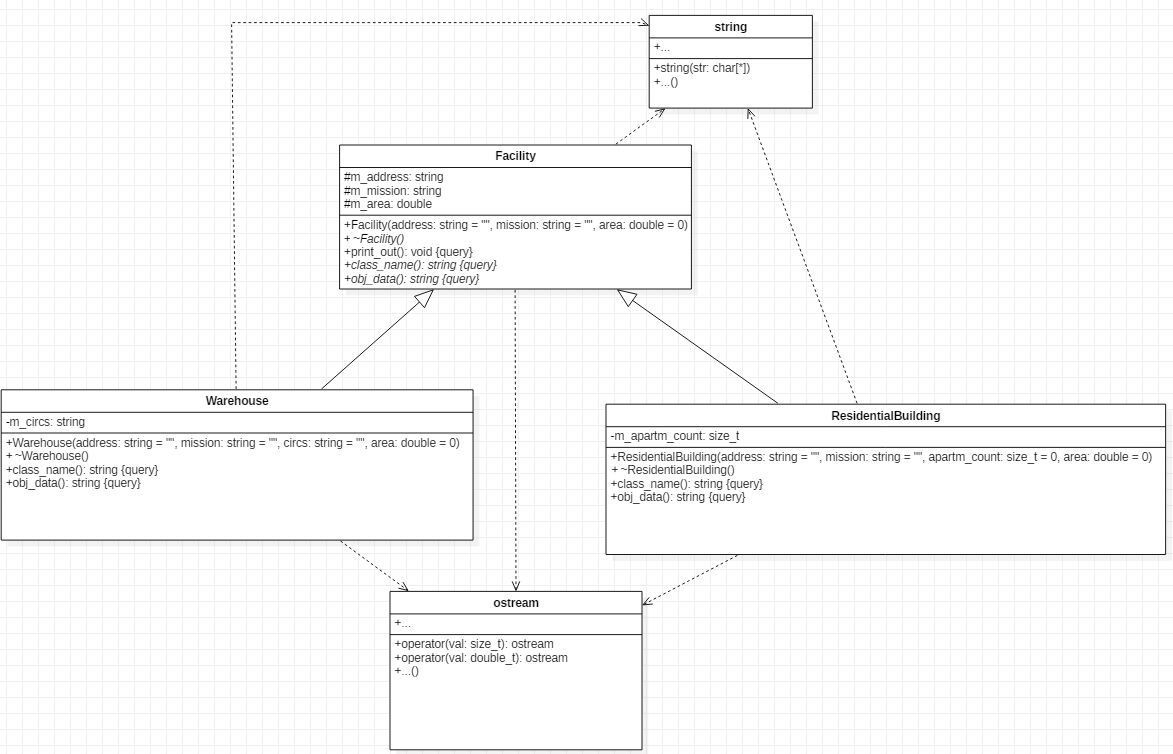
**Завдання:**

1. Вивчити теоретичні основи визначення класів, що використовують спадкування властивостей, та побудови ієрархії класів в мові С++.
2. Розробити діаграму класів в нотації UML та програмний застосунок з використанням успадкування, що реалізує ієрархію класів відповідно до свого варіанту (визначення класів та їх реалізації мають бути розташовані у файлах \*.h та \*.сpp відповідно; для ініціалізації даних класів використовувати введення з клавіатури):

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | Будівля (адреса, загальна площа, призначення) | Жилий будинок (кількість квартир) | Використовуючи віртуальні функції, не знаючи з об’єктом якого класу ви працюєте, виведіть на екран швидкість та компанію виробника для засобу пересування |
| Сховище (умови зберігання) |

1. Скласти і захистити звіт.

**UML діаграма:**

****

**Код програми:**

#pragma once

#include <string>

#include <iostream>

#include <cmath>

class **Facility**

{

public:

**Facility**(std::string address = "", std::string mission = "", double area = 0);

void **print\_out**() const;

virtual std::string ***class\_name***()const = 0 ;

virtual std::string ***obj\_data***() const = 0;

virtual ~***Facility***() = 0;

protected:

std::string m\_address, m\_mission;

double m\_area;

};

**facility.cpp:**

#include "facility.h"

Facility::**Facility**(std::string address, std::string mission, double area)

: m\_address(address), m\_mission(mission), m\_area(area)

{

}

Facility::~***Facility***()

{

}

void Facility::**print\_out**() const

{

std::cout << *class\_name*();

std::cout << *obj\_data*();

}

std::string Facility::***obj\_data***() const

{

char buffer[57];

sprintf(buffer, "address: %s\nmission: %s\narea: %f\n", m\_address.c\_str(),

m\_mission.c\_str(), m\_area);

return buffer;

}

**warehouse.h:**

#pragma once

#include "facility.h"

class **Warehouse** : public Facility

{

public:

**Warehouse**(std::string address = "", std::string mission = "",

std::string circs = "", double area = 0);

std::string ***class\_name***() const override;

std::string ***obj\_data***() const override;

~***Warehouse***() override;

private:

std::string m\_circs;

};

**warehouse.cpp:**

#include "warehouse.h"

Warehouse::**Warehouse**(std::string address, std::string mission, std::string circs, double area)

:Facility (address, mission, area), m\_circs(circs)

{

}

std::string Warehouse::***class\_name***() const

{

return "Class name: Warehouse\n";

}

std::string Warehouse::***obj\_data***() const

{

std::string base\_info(Facility::obj\_data());

char buffer[125];

sprintf(buffer, "Circumstances: %s\n", m\_circs.c\_str());

base\_info += (std::string)buffer;

return base\_info;

}

Warehouse::~***Warehouse***()

{

}

**residentialbuilding.h:**

#pragma once

#include "facility.h"

class **ResidentialBuilding** : public Facility

{

public:

**ResidentialBuilding**(std::string address = "", std::string mission = "",

size\_t apartm\_count = 0, double area = 0);

std::string ***class\_name***() const override;

std::string ***obj\_data***() const override;

~***ResidentialBuilding***() override;

private:

size\_t m\_apartm\_count;

};

**residentialbuilding.cpp:**

#include "residentialbuilding.h"

ResidentialBuilding::**ResidentialBuilding**(std::string address, std::string mission, size\_t apartm\_count,

double area)

: Facility(address, mission, area), m\_apartm\_count(apartm\_count)

{

}

std::string ResidentialBuilding::***obj\_data***() const

{

std::string base\_info(Facility::obj\_data());

char buffer[57];

sprintf(buffer, "Number of apartments: %d\n", m\_apartm\_count);

base\_info += (std::string)buffer;

return base\_info;

}

std::string ResidentialBuilding::***class\_name***() const

{

return "Class: ResidentialBuilding\n";

}

ResidentialBuilding::~***ResidentialBuilding***()

{

}

**main.cpp:**

#include <iostream>

#include "facility.h"

#include "warehouse.h"

#include "residentialbuilding.h"

#include <algorithm>

int **main**()

{

const size\_t SIZE = 5;

Facility \*\*facility\_arr = new Facility\*[SIZE];

facility\_arr[0] = new Warehouse("56783", "edsfgkjfdsj", "good", 345.4);

facility\_arr[1] = new Warehouse("11111", "1111111111", "verygood", 1111);

facility\_arr[2] = new ResidentialBuilding("351234513", "wqereqwrt", 1124, 234.345);

facility\_arr[3] = new ResidentialBuilding("22222222", "smthsmth", 12.4, 123.5);

facility\_arr[4] = new Warehouse("00000", "000000", "0good", 555.9);

std::random\_shuffle(facility\_arr, facility\_arr + SIZE);

for (size\_t i = 0; i < SIZE; ++i)

{

std::cout << "\n";

facility\_arr[i]->print\_out();

delete facility\_arr[i];

std::cout << "\n";

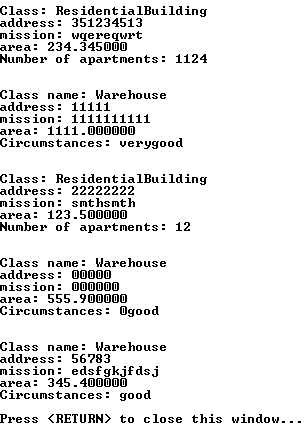
}

delete [] facility\_arr;

return 0;

}

**Результат роботи програми:**

****

**Контрольні запитання**

* 1. Навіщо потрібне перевантаження операторів?

Для того аби код виглядав більш логічно, та був зручнішим у читанні та розумінні. До того ж при роботі з об’єктами код виглядає більш компактним при використанні операторів (якщо тільки не використовувати повну версію: [obj].operator[symbol of operator]([arguments] – тоді сенсу у роботі з операторами особливо не має).

* 1. Якими двома різними способами визначаються перевантажені оператори?

Через методи класу та через friend функції.

* 1. Які операції мови С++ не можуть бути перевантажені?

Ті операції, що уможливлюють мінімальну роботу з объектами, такі як: “::”, “.”, “.\*”, “?:”, “sizeof()”, “typeid()”

* 1. Які оператори не можна перевантажити за допомогою глобальної дружньої функції?

Оператори “=”, “()”, “[]”, “->”, бо ці оператор визначені в тілі класу компілятором по замовчуванню, якщо тільки ми не перевизначимо їх у тілі класі. Якщо ж ми додаємо дружню функцію для їх перевантаження, то компілятор все одно надає стандартне перевантаження цих операторів, тому у нас виникає двоєстість – ми маємо два однакових оператори, що є помилкою.

* 1. У яких випадках операцію можна перевантажити лише за допомогою дружньої функції?

У жодному.

Nf

6. У яких випадках оператор повинен перевантажуватися дружньою функцією?

У тих випадках коли нам важливий порядок аргументів (що йдуть до оператора та після). Як у випадку з std::cout або std::cin, де об’єкт типу ostrem розміщується як правило з лівою сторони від операторів << або >>.

7. Чи обов’язково при перевантаженні операції мати параметр типу «клас» або посилання на «клас»?

Ні, якщо ми говоримо про метод класу, адже метод класу викликається через екземпляр класу та вже має доступ до данних цього об’єкту (а для унарної операції нам достатньо мати один об’єкт, для бінарної потрібно додати іще один у списку параметрів). Дружня ж функція повинна отримувати усі необхідні параметри, бо вона не є методом класу. Тому для унарної операції перевантаженої через дружню функцію ми явно повинні вказати 1 об’єкт типу «клас» (або посилання на «клас») у її параметрах. Для бінарної – 2 (один з котрих обов’язково повинен мати об’єкт типу «клас» (або посилання)).

8. Чи можна успадкувати або повторно перевантажувати у похідному класі операцію, що була перевантажена у базоваому класі?

Так.

9.Чому не можна перевантажувати деструктори?

Тому що деструктор не приймає параметрів, що є необхідною умовою для перевантаження функцій. він завжди викликається автоматично при виході відповідного об’єкту за область видимості («{}»).

10.Скільки потрібно передати параметрів функції-методу перевантаженої бінарної операції?

Один.