Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



Звіт

З лабораторної роботи №7

З дисципліни «Об’єктно-орієнтоване програмування »

На тему:

МНОЖИННЕ СПАДКУВАННЯ. ПОЛІМОРФІЗМ

Виконав: Куцевол В.С. KI-109

Прийняв: асис. каф. ЕОМ

Гузинець Н.В

Львів – 2024

**Тема роботи:** МНОЖИННЕ СПАДКУВАННЯ. ПОЛІМОРФІЗМ

**Мета.** : познайомитися із множинним спадкуванням класів та поліморфізмом.

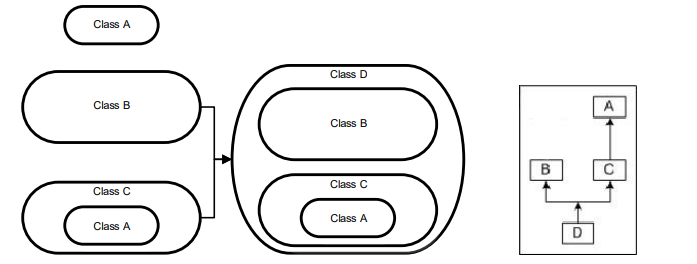
**Теоретичні відомості**

Якщо спадкування здійснюється від декількох батьківських класів одночасно, тоді воно називається множинним спадкуванням. Визначальним для похідного класу породженого множинним спадкуванням є те, що він явно чи неявно повинен успадковувати характеристики декількох базових класів.

Основні принципи одинарного спадкування, зокрема спадкування членів, модифікаторів доступу до членів базових класів, розширення та обмеження характеристик, без жодних доповнень можуть бути перенесені на множинне спадкування.

Неявним множинним спадкуванням можуть бути випадки змішаного спадкування. Результатом цих спадкуваннь є ієрархія, в якій похідний клас неявно (через один проміжний) успадкував характеристики двох базових класів

Якщо похідний клас породжується від декількох базових, то в декларації класу треба вказати усі базові класи, розділяючи їх комою, разом зі специфікаторами спадкування.



Як видно з даної схеми клас В є самостійним класом, клас А успадковується одинарно класом С, а клас D множинно успадковує класи В і С, прицьому клас D, опосередковано через клас С успадковує клас А. Тож при створенні об‘єкту класу D він міститиме в собі всі характеристики об‘єктів класів А, В і С. При цьому оскільки кожен з об‘єктів входить в об‘єкт D один раз, то ніяких конфліктів при створенні об‘єкту D не буде.

При створенні об'єкта класу, який множинно породжений, після виклику конструктора похідного класу викликатиметься конструктор найпершого батьківського класу. Якщо він є похідним від ще одного класу, то викликатиметься і буде виконуватись конструктор останнього. По закінченню роботи усіх конструкторів по гілці дерева від найпершого класу, розпочне виконуватись гілка від другого батьківського класу і т.д. Після того, як відпрацюють конструктори усіх батьківських класів, виконається тіло конструктора похідного класу.

Порядок передавання аргументів конструкторам в оголошенні конструктора похідного класу може бути довільним, оскільки виклики і виконання конструкторів визначаються порядком спадкування в оголошенні класу.

Як і у випадку одинарного спадкування при визначенні конструктора можна передавати аргументи лише конструкторам виключно базових класів. Конструкторам класів, які не є безпосередньо в ієрархії, аргументи передаються лише через визначення конструкторів проміжних класів. Наприклад, наступне визначення конструктора є неправильним, якщо O() є батьківським класом тільки класу А або B.

Порядок виклику деструкторів є таким як у конструкторів, а виконання - зворотнім. Найпершим почне виконуватись деструктор похідного класу, а далі - деструктори гілки породженої останньою в оголошенні батьківським класом. У порядку зворотному до декларації батьківських класів відпрацюють деструктори класів усіх гілок від них породжених. Лише по закінченню роботи і закритті батьківських деструкторів закриється деструктор похідного класу.

Для звертання до будь-яких членів базових класів при множинному спадкуванні як і при одинарному спадкуванні слід використати оператор (::) з іменем класу. Цей підхід є найпростішим і, очевидно, найефективнішим. Більшість логічних суперечностей, які виникають при маніпуляціях з успадкованими членами в похідних класах, розв'язуються за допомогою оператора (::).

Правило домінування, яке визначало порядок виклику успадкованих членів при простому спадкуванні, в цілому підтримується при множинному спадкуванні, але його застосування вимагає додаткових розумових зусиль. Правило домінування застосовується лише стосовно тих класів, які породжують одне одного, тобто є сусідніми (Подібно до одинарного спадкування при множинному спадкуванні здійснюється автоматичне створення зв‘язків від об'єктів нижчих класів до об'єктів вищих. Єдиною проблемою можуть стати неоднозначності батьківських зв'язків). При множинному спадкуванні цю умову треба особливо відслідковувати. В іншому випадку правило домінування не можна застосовувати і треба повертатись до використання оператора (::)

Якщо два класи породжені один від одного, то член похідного класу є домінуючим. Правило пошуку відповідного члена для виклику відбувається на стадії компіляції. У випадку, коли правило домінування не спрацьовує треба використовувати механізм (::) і явно вказувати ідентифікатор класу, якому належить даний член

Поліморфізм – це здатність коду при постійному інтерфейсі змінювати свою поведінку в залежності від ситуації, яка виникає на момент виконання. Іншими словами один і той же метод може бути визначений для об‘єктів різних класів, що є між собою в ієрархії спадкування, при цьому метод якого класу викликати вирішується під час виконання програми

Поліморфізм широко застосовується при створені складних бібліотек класів та програмних систем. Один продуманий інтерфейс може бути з успіхом використаний у різних ситуаціях, що зменшує складність програмної структури завдяки уніфікації інтерфейсу. Спрощення створення програмного 117 інтерфейсу дає можливість розробнику "тримати" в пам'яті меншу кількість інформації.

Загалом розрізняють два види поліморфізму: статичний (static або compilation-time polymorphism) і динамічний (run-time polymorphism).

Статичний поліморфізм реалізовується за допомогою так званого раннього зв’язування через механізм перевантаження функцій, методів та операторів і віртуальні класи. Він притаманний класичним структурним мовам програмування. При використанні статичного поліморфізму вибір функції чи методу, що буде викликатися здійснюється компілятором при компіляції програми (раннє зв‘язування). Вибір функції чи методу в даному випадку залежить від типу вказівника чи посилання і не залежить від типу реального об‘єкту на який вказує вказівник чи посилається посилання. Тобто при звертанні до методу об'єкту похідного класу, використовуючи вказівник чи посилання на базовий клас, викликається буде метод базового класу. Раннє зв‘язування реалізується наступним чином. Компілятор на основі вихідного коду використовує фіксовані ідентифікатори функцій і методів. На етапі компонування ці ідентифікатори замінюються фізичними адресами відповідних функцій і методів. Раннє зв'язування характеризується високою швидкістю виконання програми, оскільки єдиними витратами в період виконання є: передавання аргументів, виконання виклику функції або методу та очищення стеку. Основним недоліком раннього зв'язування є втрати гнучкості виконуваного коду

Динамічний поліморфізм реалізовується за допомогою так званого пізнього зв’язування через механізм віртуальних функцій. Він притаманний об‘єктно-орієнтованим мовам програмування і може бути застосований лише до методів класів. При використанні динамічного поліморфізму вибір методу, що буде викликатися здійснюється в процесі виконання програми (пізнє зв‘язування). Вибір методу в даному випадку залежить від типу реального об‘єкту на який вказує вказівник чи посилається посилання. Тобто при звертанні до методу об'єкту похідного класу, використовуючи вказівник чи посилання на базовий клас, викликатися буде метод похідного класу. Динамічний поліморфізм дозволяє значно підвищити ефективність і гнучкість поведінки програми, але ціною за це є сповільнення виконання програми, оскільки в процесі виконання програмі додатково необхідно приймати рішення який метод у якій ситуації викликати. Тому використання механізму пізнього зв‘язування має сенс лише за умови існування ієрархії класів. В іншому випадку єдине що ми отримаємо – це сповільнення виконання програми.

Пізнє зв‘язування при одинарному спадкуванні реалізується за допомогою таблиці віртуальних функцій (ТВФ) та вказівника vptr. Для 118 кожного класу, що містить віртуальні функції в процесі роботи програми створюється тільки одна ТВФ. Вказівник vptr в компіляторах компанії Microsoft завжди міститься за нульовим зміщенням відносно початку об‘єкта в усіх об‘єктах, що мають віртуальні функції незалежно від того чи то об‘єкти базового чи похідного класу і часто називається \_\_vfptr, а сама ТВФ називається vftable. Таблиця віртуальних функцій – це масив, кожен елемент якого містить вказівник на метод. Отримати доступ до вказівника vptr засобами мови С++ неможливо. ТВФ створюється за допомогою конструктора класу. Якщо класи знаходяться в одинарній ієрархії, то спочатку конструктор створює ТВФ для базового класу і в неї включаються всі віртуальні функції базового класу, а вказівнику vptr присвоюється адреса цієї таблиці. Віртуальні методи у ТВФ вносяться в порядку слідування, тому компілятор завжди може однозначно визначити де знаходиться той чи інший метод не зберігаючи його назви. Далі викликається конструктор для похідного класу і створюється копія ТВФ базового класу в якій при необхідності змінюються адреси перевизначених віртуальних методів та додаються нові знову ж таки в порядку слідування починаючи з індекса на якому закінчилася індексація віртуальних методів базового класу, а vptr присвоюється адреса цієї ТВФ. Процес продовжується доти, поки не створиться останній об‘єкт похідного класу. При виклику віртуального методу компілятор генерує код звернення за вказівником vptr до ТВФ, за допомогою черговості оголошення методу визначає індекс методу в ТВФ і в необхідне місце програми вставляє код непрямого виклику методу за адресою, що міститься в ТВФ за відповідним зміщенням (у відповідній комірці масиву, який реалізує ТВФ).

При знищені об‘єкту процес відбувається у зворотньому порядку відносно створення об‘єкту – спочатку знищується об‘єкт похідного класу, а vptr присвоюється адреса об‘єкту на рівень вище в ієрархії класів і т.д. поки не знищиться найвищий в ієрархії об‘єкт класу.

У випадку якщо використовується множинне спадкування, то для кожного класу, що містить множинне спадкування додаються стільки вказівників vptr, скільки класів спадкує похідний класу. Це робиться для того, щоб кожен з базових класів можна було використати в якості вказівника чи посилання

Мова С++ є гібридною мовою програмування в плані підтримки поліморфізму, оскільки в ній реалізована підтримка як статичного так і динамічного поліморфізму. Тож відповідно до типу зв'язування в С++ розрізняють звичайні функції і методи – функції і методи раннього зв'язування і поліморфні або віртуальні методи (virtual methods) - методи пізнього зв'язування

Для того, щоб оголосити віртуальний метод треба перед оголошенням методу поставити ключове слово virtual. Для того, щоб метод став віртуальним достатньо вжити ключове слово virtual лише раз у всій ієрархії спадкування. Зазвичай це робиться у базовому класі, а в похідних класах цей метод перевизначається як звичайний метод класу.

**Хід виконання роботи**

**Варіант 12**

Завдання : Спроектувати і реалізувати ієрархію класів, що описують предметну область згідно варіанту, яка реалізується класом 1. Клас 1 в свою чергу утворюється шляхом множинного спадкування класів 2 і 3 кожен з яких в свою чергу успадковує клас 4.

Додаткові вимоги:

1. Базовий клас містить мінімум один віртуальний метод, один невіртуальний метод і одну динамічно створювану властивість.

2. Забезпечити механізми коректної роботи конструкторів і деструкторів.

3. Перевантажити оператор присвоєня з метою його коректної роботи.

4. Кожен з класів має містити мінімум одину властивість і 4 методи.

5. Написати main() функцію де створити об‘єкт класу 1 і продемонструвати різницю між статичним і динамічним поліморфізмом.



//file main.cpp

#include <iostream>

#include "CDevice.h"

#include"CPhone.h";

#include"CComputer.h"

#include"CSmartphone.h"

using std::cout;

using std::cin;

using std::endl;

int main() {

CDevice device;

CPhone phone;

CComputer computer;

CSmartphone smartphone;

cout << "--------------------------------------------------------------" << endl;

cout << "Reprezentation of CDevice class:" << endl;

device.SetBatteryCapacity(5000);

device.SetName("Device 1");

device.AddModelNumber(" V0.1");

cout << "Devise battery capacity is: " << device.GetBatteryCapacity() << endl

<< "Device name is: " << device.GetName() << endl << endl;

cout << "--------------------------------------------------------------" << endl;

cout << "Reprezentation of CPhone class:" << endl;

phone.SetName("Phone 1");

phone.SetAudioPower(500);

phone.SetBatteryCapacity(5500);

phone.SetScreenDiagonal(4.4);

phone.AddModelNumber(" V 1.0");

cout << "Phone`s audio power is : " << phone.GetAudioPower() << endl

<< "Phone`s battery capacity is: " << phone.GetBatteryCapacity() << endl

<< "Phone`s name is: " << phone.GetName() << endl

<< "Phone`s screen diagonal is: " << phone.GetScreenDiagonal() << endl << endl;

phone.MakeCall();

phone.SendMessage();

phone.ShowPhoto();

phone.ConnectSpeaker();

cout << "Phone`s audio power is : " << phone.GetAudioPower() << endl;

cout << "--------------------------------------------------------------" << endl;

cout << "Reprezentation of CComputer class:" << endl;

computer.SetBatteryCapacity(55000);

computer.SetComputerPerformance(5.4);

computer.SetName("Computer 1");

computer.SetProcessorName("Core 1");

computer.AddModelNumber(" V 1.1");

cout << "Computer`s battery capacity: " << computer.GetBatteryCapacity() << endl

<< "Computer`s performance: " << computer.GetComputerPerformance() << endl

<< "Computer`s name: " << computer.GetName() << endl

<< "Compuert`s processor name: " << computer.GetProcessorName() << endl << endl;

computer.calculate(1, 2);

computer.calculate();

computer.OverclockProcessor();

computer.AddProcessorModelNumber(" I9 1488 SS");

cout << endl << "Computer`s performance: " << computer.GetComputerPerformance() << endl;

cout << "Compuert`s processor name: " << computer.GetProcessorName() << endl;

cout << "--------------------------------------------------------------" << endl;

cout << "Reprezentation of CSmartphone class:" << endl;

smartphone.SetAudioPower(500);

smartphone.SetBatteryCapacity(4000);

smartphone.SetComputerPerformance(3.2);

smartphone.SetIsWaterproof(true);

smartphone.SetName("Smartphone 1");

smartphone.SetPrice(1000);

smartphone.SetProcessorName("Snapfire");

smartphone.SetScreenDiagonal(5.5);

cout << "Smartphone`s audio power: " << smartphone.GetAudioPower() << endl

<< "Smartphones`s battery capacity: " << smartphone.GetBatteryCapacity() << endl

<< "Smartphone`s performance: " << smartphone.GetComputerPerformance() << endl

<< "Smartphone`s waterproof: " << smartphone.GetIsWaterProof() << endl

<< "Smartphone`s name: " << smartphone.GetName() << endl

<< "Smartphone`s processor name: " << smartphone.GetProcessorName() << endl

<< "Smartphone`s screen diagonal: " << smartphone.GetScreenDiagonal() << endl

<< "Smartphone`s price: " << smartphone.GetPrice() << endl << endl;

smartphone.AddModelNumber(" V2");

smartphone.MakeCall();

smartphone.SendMessage();

smartphone.ShowPhoto();

smartphone.ConnectSpeaker();

smartphone.calculate(3, 4);

smartphone.calculate();

smartphone.OverclockProcessor();

smartphone.AddProcessorModelNumber(" Cookie");

smartphone.UnlockPhone();

smartphone.LockPhone();

smartphone.TurnOff();

smartphone.PlayGame();

cout << endl <<"Smartphone`s audio power:" << smartphone.GetAudioPower() << endl

<< "Smartphones`s battery capacity:" << smartphone.GetBatteryCapacity() << endl

<< "Smartphone`s performance" << smartphone.GetComputerPerformance() << endl

<< "Smartphone`s waterproof:" << smartphone.GetIsWaterProof() << endl

<< "Smartphone`s name: " << smartphone.GetName() << endl

<< "Smartphone`s processor name:" << smartphone.GetProcessorName() << endl

<< "Smartphone`s screen diagonal:" << smartphone.GetScreenDiagonal() << endl

<< "Smartphone`s price:" << smartphone.GetPrice() << endl;

return 0;

}

//file CDevice.h

#pragma once

#include <iostream>

#include <string>

using std::string;

class CDevice

{

private:

string name;

int batteryCapacity;

public:

void SetName(string);

string GetName();

void SetBatteryCapacity(int);

int GetBatteryCapacity();

virtual void PlayGame() {};

void AddModelNumber(string);

CDevice();

virtual ~CDevice() {};

};

//file CDevice.cpp

#include "CDevice.h"

void CDevice::SetName(string name) {

CDevice::name = name;

}

string CDevice::GetName() {

return CDevice::name;

}

void CDevice::SetBatteryCapacity(int batteryCapacity) {

CDevice::batteryCapacity = batteryCapacity;

}

int CDevice::GetBatteryCapacity() {

return CDevice::batteryCapacity;

}

void CDevice::AddModelNumber(string modelNumber) {

CDevice::name.append(modelNumber);

}

CDevice::CDevice() {

CDevice::name = "Device 0";

CDevice::batteryCapacity = 0;

}

//file CPhone.h

#pragma once

#include "CDevice.h"

class CPhone :

public virtual CDevice

{

private:

float screenDiagonal;

int audioPower;

public:

void SetScreenDiagonal(float);

float GetScreenDiagonal();

void SetAudioPower(int);

int GetAudioPower();

void MakeCall();

void SendMessage();

void ShowPhoto();

void ConnectSpeaker();

CPhone();

};

//file CPhone.cpp

#include "CPhone.h"

#include <iostream>

#include <string>

using std::cout;

using std::cin;

using std::endl;

using std::string;

void CPhone::SetScreenDiagonal(float screenDiagonal) {

CPhone::screenDiagonal = screenDiagonal;

}

float CPhone::GetScreenDiagonal() {

return CPhone::screenDiagonal;

}

void CPhone::SetAudioPower(int audioPower) {

CPhone::audioPower = audioPower;

}

int CPhone::GetAudioPower() {

return CPhone::audioPower;

}

void CPhone::MakeCall() {

cout << "Ring...Ring...Ring" << endl;

}

void CPhone::SendMessage() {

string str1;

cout << "Enter message: ";

cin >> str1;

cout << "Your message have been sent" << endl;

}

void CPhone::ShowPhoto() {

cout << "Here is your photo: " << endl;

cout << endl << "#############################################################" << endl;

cout << "# \_ #" << endl;

cout << "# -=\\`\\ #" << endl;

cout << "# |\\ \_\_\_\_\\\_\\\_\_ #" << endl;

cout << "# -=\\c`\"\"\"\"\"\"\" \"`) #" << endl;

cout << "# `~~~~~/ /~~`\ #" << endl;

cout << "# -==/ / #" << endl;

cout << "# '-' #" << endl;

cout << "# \_ \_ #" << endl;

cout << "# ( ` )\_ #" << endl;

cout << "# ( ) `) #" << endl;

cout << "# (\_ (\_ . \_) \_) #" << endl;

cout << "# \_ #" << endl;

cout << "# ( ) #" << endl;

cout << "# \_ . ( ` ) . ) #" << endl;

cout << "# ( \_ )\_ (\_, \_( ,\_)\_) #" << endl;

cout << "# (\_ \_(\_ ,) #" << endl;

cout << "#############################################################" << endl;

}

void CPhone::ConnectSpeaker() {

int speakersPower;

cout << "Enter speakers power: ";

cin >> speakersPower;

CPhone::SetAudioPower(CPhone::audioPower + speakersPower);

cout << "Speaker have been connected" << endl;

}

CPhone::CPhone() :

CDevice::CDevice() {

CPhone::screenDiagonal = 0;

CPhone::audioPower = 0;

};

//file CComputer.h

#pragma once

#include "CDevice.h"

class CComputer :

public virtual CDevice

{

private:

float computerPerformance;

string processorName;

public:

void SetComputerPerformance(float);

float GetComputerPerformance();

void SetProcessorName(string);

string GetProcessorName();

void calculate();

void calculate(int, int);

void OverclockProcessor();

void AddProcessorModelNumber(string);

CComputer();

};

//file CComputer.cpp

#include "CComputer.h"

#include <iostream>

#include <cstdlib>

#include <string>

using std::cout;

using std::cin;

using std::endl;

using std::string;

void CComputer::SetComputerPerformance(float computerPerformence) {

CComputer::computerPerformance = computerPerformence;

}

float CComputer::GetComputerPerformance() {

return CComputer::computerPerformance;

}

void CComputer::SetProcessorName(string processorName) {

CComputer::processorName = processorName;

}

string CComputer::GetProcessorName() {

return CComputer::processorName;

}

void CComputer::calculate() {

int number1, number2;

cout << "Enter first number (int): " << endl;

cin >> number1;

cout << "Enter second number (int): " << endl;

cin >> number2;

cout << "Sum is: " << number1 + number2 << endl;

}

void CComputer::calculate(int number1, int number2) {

cout << "Sum of entered 2 numbers is " << number1 + number2 << endl;

}

void CComputer::OverclockProcessor() {

std::srand(time(0));

int overlockingModifier = (std::rand() % 2 + 1);

CComputer::SetComputerPerformance(CComputer::computerPerformance + overlockingModifier);

cout << "Processor have beed overlocked" << endl;

}

void CComputer::AddProcessorModelNumber(string modelNumber) {

string processorName = CComputer::GetProcessorName();

CComputer::SetProcessorName(CComputer::processorName + modelNumber);

}

CComputer::CComputer() :

CDevice() {

CComputer::computerPerformance = 1;

CComputer::processorName = "Processor 0";

}

//file CSmartphone.h

#pragma once

#include"CPhone.h"

#include "CComputer.h"

#include <iostream>

using std::cout;

using std::cin;

using std::endl;

class CSmartphone :

public CPhone,

public CComputer

{

private:

float price;

bool isWaterproof;

public:

void SetPrice(float);

float GetPrice();

void SetIsWaterproof(bool);

bool GetIsWaterProof();

void UnlockPhone();

void LockPhone();

void TurnOff();

void PlayGame() {

int userChoice;

std::srand(time(0));

int randomNumber = (std::rand() % 10 + 1);

cout << "Guess a number between 1 and 10" << endl;

cin >> userChoice;

if (userChoice == randomNumber) {

cout << "You win !" << endl;

}

else {

cout << "You lost , that was " << randomNumber << endl;

}

}

CSmartphone();

};

//file CSmartphone.cpp

#include "CSmartphone.h"

#include <iostream>

using std::cout;

using std::cin;

using std::endl;

void CSmartphone::SetPrice(float price) {

CSmartphone::price = price;

}

float CSmartphone::GetPrice() {

return CSmartphone::price;

}

void CSmartphone::SetIsWaterproof(bool isWaterproof) {

CSmartphone::isWaterproof = isWaterproof;

}

bool CSmartphone::GetIsWaterProof() {

return CSmartphone::isWaterproof;

}

void CSmartphone::UnlockPhone() {

cout << "Phone is unlocked" << endl;

}

void CSmartphone::LockPhone() {

cout << "Phone is locked" << endl;

}

void CSmartphone::TurnOff() {

cout << "Phone is turned off" << endl;

}

CSmartphone::CSmartphone() :

CComputer::CComputer(), CPhone::CPhone() {

CSmartphone::price = 1;

CSmartphone::isWaterproof = true;

}

void PlayGame(){

int userChoice;

int randomNumber = (std::rand() % 10 + 1);

cout << "Guess a number between 1 and 10" << endl;

cin >> userChoice;

if (userChoice == randomNumber) {

cout << "You win !";

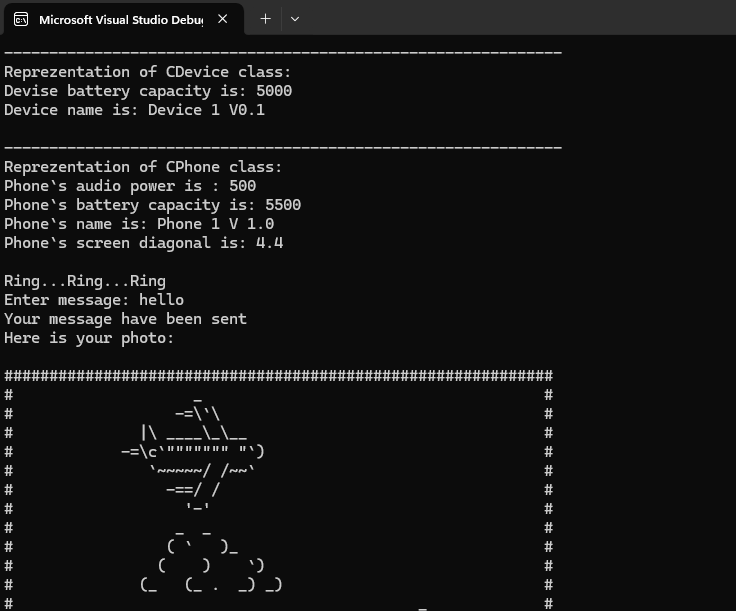
}

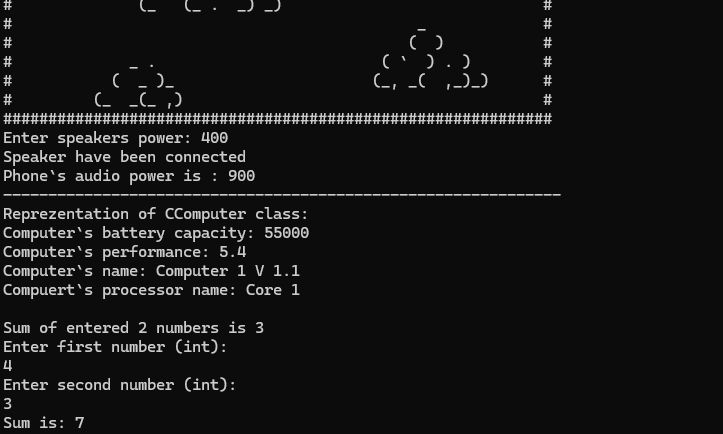
else {

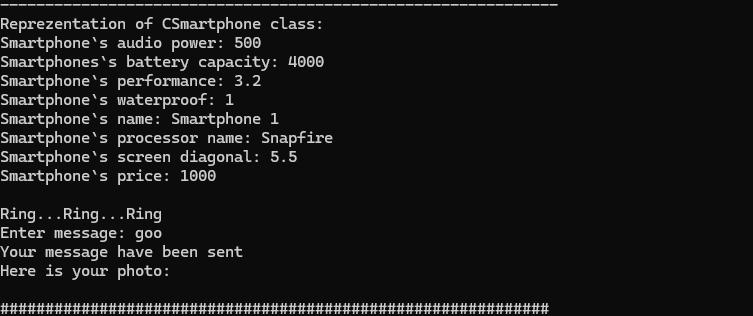
cout << "You lost , that was " << randomNumber;

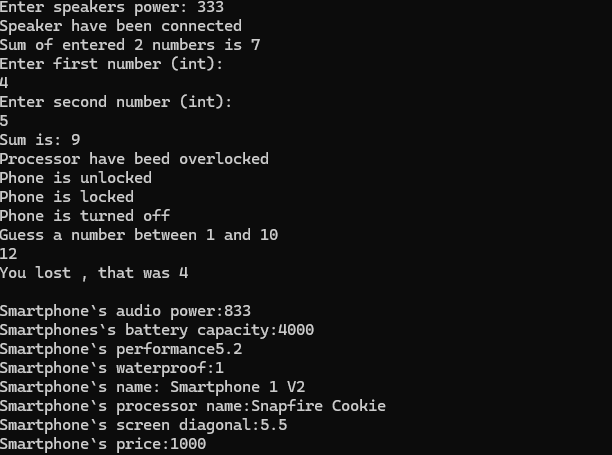
}

}









**Висновок**

Під час цієї лабораторної роботи я вивчив концепції множинного спадкування та поліморфізму в мові програмування C++. Множинне спадкування дозволяє класу успадковувати властивості та методи з більш ніж одного батьківського класу, тоді як поліморфізм дозволяє використовувати об'єкти підкласів через їх базові класи, забезпечуючи гнучкість та розширюваність коду.

Під час цієї роботи я досліджував, як коректно використовувати множинне спадкування для створення ієрархій класів, а також як використовувати поліморфізм для заміни методів базових класів у підкласах. Я вивчив важливі поняття, такі як віртуальні функції та ключове слово override, які допомагають реалізувати поліморфізм у C++.

Ця лабораторна робота дала мені можливість глибше зрозуміти принципи роботи з множинним спадкуванням та поліморфізмом. Я вважаю, що ці знання будуть корисні у моїй подальшій роботі над проектами, оскільки дозволять мені створювати більш складні та гнучкі програми з допомогою об'єктно-орієнтованого підходу.

Top of Form