Trắc nghiệm TASK 8

	0	1	2	3	4	5	6	7	8	9
0	_	A	A	B	A	A	A	A	A	A
1	A	A	A	A	A	A	A	A	A	A
2	A	A	A	A	A	A	A	A	A	A
3	A	A	A	A	A	A	A	A	A	A
4	A	A	B	A	A	A	A	A	A	A
5	A	A	A	A	A	A	A	A	A	A
6	A	A								

Phần đọc code:

Phần đọc code TASK 8

Câu 1: Kết quả và giải thích

```
Balance: 50
Insufficient funds!
Balance: 50
```

```
// GIẢI THÍCH
BankAccount acc(100); // Khởi tạo tài khoản ngân hàng, vì 100 >= 0 nên được gán
trực tiếp vào balance
acc.withdraw(50); // Vì 50(amout) > 0 && 50(amount) < 100(balance) == 1 rút ra
thành công
acc.withdraw(60); // Vì 60(amout) > 0 && 60(amount) < 50(balance) == 0 rút ra
không thành công
```

Câu 2: Kết quả và giải thích

```
Báo lỗi biên dịch.
```

```
// GIẢI THÍCH
Vì privateVar là biến private, sau khi thừa kế vẫn là private, nên class Derived
không truy xuất trực tiếp được privateVar trong hàm accessVariables()
```

Câu 3: Kết quả và giải thích

```
Area: 20
```

```
// GIẢI THÍCH
Rectangle r(4, 5); // Gán width = 4, height = 5
printArea(r); // Vì printArea là hàm bạn nên có quyền truy cập tất cả thuộc tính và phương thức
```

Câu 4: Kết quả và giải thích

```
Car Model: Tesla Model S
Horsepower: 670
```

```
// GIẢI THÍCH
Car myCar("Tesla Model S", 670); // Khai báo object myCar
Engine e; // Khai báo object e, là class bạn với class Car nên có quyền truy cập thuộc tính của class Car
```

> Câu 5: Kết quả và giải thích

```
Animal makes a sound
Dog barks
```

```
// GIẢI THÍCH
Dog d; // Vì khai báo lớp Dog là con của Animal nên có quyền truy cập phương thức chung của Animal và riêng của Dog
```

> Câu 6: Kết quả và giải thích

```
Grandparent class
Parent class
Child class
```

```
// GIẢI THÍCH
Child c; // class child kế thừa public từ Parent, nên các thuộc tính và phương thức của Parent ban đầu sẽ có mức độ truy cập như cũ, tương tự cha với lớp Grandparent
c.show(); // Gọi từ Grandparent
```

```
c.display(); // Gọi từ Parent
c.print(); // Gọi từ Child
```

Câu 7: Kết quả và giải thích

```
Engine started
Wheels rolling
Car is driving
```

```
// GIẢI THÍCH
// Đây là kiểu kế thừa nhiều lớp (Multiple inheritance)
// Việc truy xuất thuộc tính, phương thức sẽ giống như kế thừa một lớp trong
trường hợp không có sự xuất hiện trùng phương thức, thuộc tính
```

Câu 10: Kết quả và giải thích

```
Lỗi biên dịch
```

```
// GIẢI THÍCH
cout << "Private: " << privateVar << endl; // 17
// Kế thừa public nên các thuộc tính private, protected, public vẫn giữ như cũ so
với lớp DerivedPublic nên lớp này không có quyền truy xuất thuộc tính private
trong lớp Base
```

Câu 11: Kết quả và giải thích

```
Lỗi biên dịch
```

```
// GIẢI THÍCH
// Kế thừa protected nên các thuộc tính private, protected, public sẽ bị đổi mức
truy cập đối với lớp kế thừa là private, protected, protected -> protected cho
phép truy xuất giữa các lớp con với các thuộc tính public, protected của lớp cha
cout << d.publicVar; // Không có quyền truy cập thuộc tính publicVar đã bị chuyển
thành protected.
```

Câu 12: Kết quả và giải thích

```
Lỗi biên dịch
```

```
// GIẢI THÍCH
cout << d.publicVar; // Tương tự không có quyền truy cập thuộc tính publicVar đã bị chuyển thành private.
```

Câu 13: Kết quả và giải thích

```
Lỗi biên dịch
```

```
// GIẢI THÍCH
cout << privateVar; // 14 // Không có quyền truy cập thuộc tính private privateVar trong class thừa kế.
```

Câu 14: Kết quả và giải thích

```
Lỗi biên dịch
```

```
// GIẢI THÍCH
cout << pub.protectedVar << endl; // Lỗi truy xuất thuộc tính
cout << prot.publicVar << endl; // Lỗi truy xuất thuộc tính
cout << priv.publicVar << endl; // Lỗi truy xuất thuộc tính
```

Câu 15: Kết quả và giải thích

```
Base constructor called!
Derived constructor called!
```

```
// GIẢI THÍCH
Derived* d = new Derived(); // Khi khai báo constructor của cha sẽ được gọi trước con
```

Câu 16: Kết quả và giải thích

```
Base constructor called!
Derived constructor called!
Derived destructor called!
Base destructor called!
```

```
// GIẢI THÍCH
Derived* d = new Derived(); // Khi khai báo constructor của cha sẽ được gọi trước khi gọi constructor của cha
delete d; // Việc huỷ ngược lại với khai báo khi lớp con được gọi huỷ trước khi lớp cha bị huỷ
```

Câu 17: Kết quả và giải thích

```
Base constructor called with value: 10
Derived constructor called with value: 10
```

```
// GIẢI THÍCH
Derived d(10); // Constructor lớp cha sẽ được gọi trước, gán value = 10, và vì thuộc tính protected sau khi kế thừa public vẫn là protected nên lớp con có quyền truy xuất
```

Câu 20: Kết quả và giải thích

```
2 integers: 5
2 floats: 6
3 integers: 6
```

```
// GIẢI THÍCH
// Nạp chồng phương thức (Function Overloading)
// Có thể có những phương thức cùng tên trong cùng một lớp, chúng có danh sách tham số khác nhau (số lượng tham số hoặc kiểu dữ liệu của tham số).
// Trong ví dụ này, phương thức add được nạp chồng để xử lý các trường hợp khác nhau của phép cộng.
```

Câu 19: Kết quả và giải thích

```
Enter size of vectors: 3
Enter elements for Vector 1: 1 2 3
Enter elements for Vector 2: 1 2 3
Vector 1: [1, 2, 3]
Vector 2: [1, 2, 3]
Vectors are equal!
---------------------------------
Enter size of vectors: 3
Enter elements for Vector 1: 1 2 3
```

```
Enter elements for Vector 2: 1 2 4
Vector 1: [1, 2, 3]
Vector 2: [1, 2, 4]
Vectors are not equal!
```

```
// GIẢI THÍCH
// Toán tử so sánh == được nạp chồng để so sánh hai đối tượng Vector
// Toán tử so sánh != được nạp chồng dựa trên toán tử ==
// Toán tử << được nạp chồng để in ra các phần tử của Vector
// Toán tử >> được nạp chồng để nhập các phần tử cho Vector
```

Bài tập

Câu 1:

```
#include <iostream>
using namespace std;

// Lớp cha Animal
class Animal {
protected:
    string name;
public:
    Animal(string n) : name(n) {
        cout << "Animal created: " << name << endl;
    }
};

// Lớp Mammal kế thừa từ Animal
class Mammal : public Animal {
protected:
    bool hasFur;
public:
    Mammal(string n, bool fur) : Animal(n), hasFur(fur) {
        cout << "Mammal created: " << name << ", Has Fur: " << (hasFur ? "Yes" :
"No") << endl;
    }
};

// Lớp Bird kế thừa từ Animal
class Bird : public Animal {
protected:
    bool canFly;
public:
    Bird(string n, bool fly) : Animal(n), canFly(fly) {
        cout << "Bird created: " << name << ", Can Fly: " << (canFly ? "Yes" :
"No") << endl;
    }
};
```

```
// Lớp Dog kế thừa từ Mammal
class Dog : public Mammal {
private:
    string breed;
public:
    Dog(string n, string b) : Mammal(n, true), breed(b) {
        cout << "Dog created: " << name << ", Breed: " << breed << endl;
    }
};

// Lớp Duck kế thừa từ Bird
class Duck : public Bird {
private:
    bool isDomestic;
public:
    Duck(string n, bool fly, bool domestic) : Bird(n, fly), isDomestic(domestic) {
        cout << "Duck created: " << name << ", Domestic: " << (isDomestic ? "Yes"
: "No") << endl;
    }
};

int main() {
    Dog d("Buddy", "Golden Retriever");
    Duck dk("Daffy", true, false);
    return 0;
}
```

Câu 2:

```
#include <iostream>
using namespace std;

class AIModel {
protected:
    string modelName;
public:
    AIModel(string name) : modelName(name) {
        cout << "AIModel: " << modelName << " created." << endl;
    }
};

class MachineLearningModel : public AIModel {
protected:
    string algorithm;
public:
    MachineLearningModel(string name, string algo) : AIModel(name),
algorithm(algo) {
        cout << "MachineLearningModel using " << algorithm << " created." << endl;
    }
};
```

```
class DeepLearningModel : public MachineLearningModel {
protected:
    int numLayers;
public:
    DeepLearningModel(string name, string algo, int layers) :
MachineLearningModel(name, algo), numLayers(layers) {
        cout << "DeepLearningModel with " << numLayers << " layers created." <<
endl;
    }
};

class NeuralNetwork : public DeepLearningModel {
protected:
    int numNeurons;
public:
    NeuralNetwork(string name, string algo, int layers, int neurons) :
DeepLearningModel(name, algo, layers), numNeurons(neurons) {
        cout << "NeuralNetwork with " << numNeurons << " neurons created." <<
endl;
    }
};

class CNN : public NeuralNetwork {
protected:
    int numFilters;
public:
    CNN(string name, string algo, int layers, int neurons, int filters) :
NeuralNetwork(name, algo, layers, neurons), numFilters(filters) {
        cout << "CNN with " << numFilters << " filters created." << endl;
    }
};

class RNN : public NeuralNetwork {
protected:
    int timeSteps;
public:
    RNN(string name, string algo, int layers, int neurons, int steps) :
NeuralNetwork(name, algo, layers, neurons), timeSteps(steps) {
        cout << "RNN with " << timeSteps << " time steps created." << endl;
    }
};

int main() {
    CNN cnn("ImageClassifier", "Convolutional", 10, 1000, 32);
    RNN rnn("TextGenerator", "Recurrent", 5, 500, 20);
    return 0;
}
```

Câu 3:

```cpp
#include <iostream>
using namespace std;

class Entity {
protected:
    string name;
public:
    Entity(string n) : name(n) {
        cout << "Entity created: " << name << endl;
    }
};

class Movable : virtual public Entity {
protected:
    float speed;
public:
    Movable(string n, float spd) : Entity(n), speed(spd) {
        cout << name << " is movable with speed: " << speed << endl;
    }
    void move() {
        cout << name << " is moving at speed " << speed << endl;
    }
};

class Attackable : virtual public Entity {
protected:
    int attackPower;
    int health;
public:
    Attackable(string n, int atk, int hp) : Entity(n), attackPower(atk),
health(hp) {
        cout << name << " is attackable with power: " << attackPower << " and
health: " << health << endl;
    }
    void attack() {
        cout << name << " attacks with power " << attackPower << endl;
    }
};

class Player : public Movable, public Attackable {
private:
    int level;
public:
    Player(string n, float spd, int atk, int hp, int lvl)
        : Entity(n), Movable(n, spd), Attackable(n, atk, hp), level(lvl) {
        cout << name << " is a player at level: " << level << endl;
    }
    void levelUp() {
        level++;
        cout << name << " leveled up to: " << level << endl;
    }
};
```

```
int main() {
    Player p("Hero", 5.0f, 20, 100, 1);
    p.move();
    p.attack();
    p.levelUp();
    return 0;
}
```