1.Ambiguity in different Argument

#include <iostream>

using namespace std;

void f1(int x = 0)

{

cout << "f1 with one argument: " << x <<endl;

}

void f1(int x, int y = 0)

{

cout << "f1 with two arguments: "<< x << "," << y <<endl;

}

int main()

{

int x=5;

f1(x);

return 0;

}

2.Ambiguty in multiple inheritance

#include<iostream>

using namespace std;

class A

{

public:

void show()

{

cout<<"This is class A"<<endl;

}

};

class B

{

public:

void show()

{

cout<<"This is class B"<<endl;

}

};

class C:public A,public B

{

};

int main()

{

C obj;

obj.show;

return 0;

}

3.

#include <iostream>

#include <stdexcept>

using namespace std;

class Distance {

private:

int feet;

int inches;

void method() {

if (inches >= 12) {

feet += inches / 12;

inches %= 12;

} else if (inches < 0) {

feet -= (-inches) / 12 + 1;

inches = 12 - (-inches) % 12;

}

if (feet < 0 && inches != 0) {

feet -= 1;

inches = 12 - inches;

}

}

public:

Distance(int ft = 0, int in = 0) : feet(ft), inches(in) {

method();

}

void display() const

{

cout << feet << " feet " << inches << " inches" <<endl;

}

Distance operator+(const Distance& other) const

{

return Distance(feet + other.feet, inches + other.inches);

}

Distance operator-(const Distance& other) const

{

return Distance(feet - other.feet, inches - other.inches);

}

bool operator==(const Distance& other) const {

return (feet == other.feet) && (inches == other.inches);

}

bool operator!=(const Distance& other) const

{

return !(\*this == other);

}

bool operator<(const Distance& other) const {

if (feet < other.feet) return true;

if (feet == other.feet && inches < other.inches) return true;

return false;

}

bool operator>(const Distance& other) const {

return other < \*this;

}

bool operator<=(const Distance& other) const {

return !(\*this > other);

}

bool operator>=(const Distance& other) const {

return !(\*this < other);

}

};

int main() {

Distance d1(5, 8);

Distance d2(3, 4);

Distance d3(5, 10);

cout << "d1: "; d1.display();

cout << "d2: "; d2.display();

cout << "d3: "; d3.display();

Distance d\_add = d1 + d2;

cout << "d1 + d2: "; d\_add.display();

Distance d\_sub = d1 - d2;

cout << "d1 - d2: "; d\_sub.display();

cout << "d1 == d2: " << (d1 == d2) << std::endl;

cout << "d1 != d2: " << (d1 != d2) << std::endl;

cout << "d1 < d3: " << (d1 < d3) << std::endl;

cout << "d1 > d2: " << (d1 > d2) << std::endl;

cout << "d1 <= d1: " << (d1 <= d1) << std::endl;

cout << "d2 >= d3: " << (d2 >= d3) << std::endl;

return 0;

}

4.

#include<iostream>

using namespace std;

class A

{

public:

void show()

{

cout<<"This is class A"<<endl;

}

};

class B

{

public:

void show()

{

cout<<"This is class B"<<endl;

}

};

class C:public A,public B

{

};

int main()

{

C obj;

obj.show;

return 0;

}

5.

#include <iostream>

using namespace std;

class Animal

{

public:

void eat()

{

cout<<"eating";

}

};

class dog:public Animal

{

public:

void eat()

{

cout<<"Eating bread";

}

};

int main()

{

dog d=dog();

d.eat();

d.Animal::eat();

return 0;

}

7.

#include <iostream>

using namespace std;

class A

{

int x=5;

public:

void display()

{

cout<<"value of x is"<<x<<endl;

}

};

class B:public A

{

int y=10;

public:

void display()

{

cout<<"value of y is"<<y<<endl;

}

};

int main()

{

A \*a;

B b;

a= &b;

a->display();

return 0;

}

8.

#include <iostream>

using namespace std;

class Shape {

public:

virtual void draw() const = 0;

virtual ~Shape() {}

};

class Square : public Shape {

public:

void draw() const override {

cout << "\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*" << endl;

}

};

class Circle : public Shape {

public:

void draw() const override {

cout << " OOO " << endl;

cout << "O O" << endl;

cout << " OOO " << endl;

}

};

class Triangle : public Shape {

public:

void draw() const override {

cout << " \* " << endl;

cout << " \* \* " << endl;

cout << "\*\*\*\*\*" << endl;

}

};

void printShape(const Shape\* shape) {

shape->draw();

}

int main() {

Shape\* shapes[3];

shapes[0] = new Square();

shapes[1] = new Circle();

shapes[2] = new Triangle();

for (int i = 0; i < 3; ++i) {

printShape(shapes[i]);

delete shapes[i];

}

return 0;

}

9.

#include <iostream>

#include <string>

using namespace std;

class Animal {

public:

virtual string makeSound() const = 0;

virtual ~Animal() {}

};

class Dog : public Animal {

public:

string makeSound() const override {

return "Woof!";

}

};

class Cat : public Animal {

public:

string makeSound() const override {

return "Meow!";

}

};

class Bird : public Animal {

public:

string makeSound() const override {

return "Chirp!";

}

};

void playAnimalSound(const Animal\* animal) {

cout << animal->makeSound() << endl;

}

int main() {

Animal\* animals[3];

animals[0] = new Dog();

animals[1] = new Cat();

animals[2] = new Bird();

for (int i = 0; i < 3; ++i) {

playAnimalSound(animals[i]);

delete animals[i];

}

return 0;

}