Οι λέξεις-κλειδιά **`public`**, **`private`**, **`protected`**, **`friend`**, και **`virtual`** είναι βασικά εργαλεία για τον καθορισμό της **προσβασιμότητας**, της **κληρονομικότητας** και του **πολυμορφισμού** στη C++.

`public`, `private`, `protected`

Αυτές καθορίζουν το επίπεδο προσβασιμότητας των μελών (μεταβλητών και συναρτήσεων) μιας κλάσης.

Modifier	Προσβάσιμο από την ίδια την κλάση	Από υποκλάσεις	Από αντικείμενα (έξω απ' την κλάση)
`public`	✓	✓	✓
`protected`	✓	✓	✗
`private`	✓	✗	✗

Παράδειγμα:

```
class MyClass {
public:
    int a;         // Προσβάσιμο παντού
protected:
    int b;         // Προσβάσιμο μόνο από υποκλάσεις
private:
    int c;         // Μόνο εντός της MyClass
};
```

`friend`

Η `friend` συνάρτηση ή κλάση μπορεί να έχει πρόσβαση ακόμα και στα `private` και `protected` μέλη της κλάσης στην οποία δηλώνεται ως φίλη.

Παράδειγμα:

```
class MyClass {
private:
    int secret = 42;

    // Δήλωση φιλικής συνάρτησης
    friend void showSecret(MyClass obj);
};

void showSecret(MyClass obj) {
    cout << obj.secret << endl;  // Μπορεί να δει το private!
}
```

`virtual`

Η `virtual` δηλώνει μια μέθοδο ως **εικονική**, που σημαίνει ότι μπορεί να **γίνει override** σε υποκλάσεις και να υποστηρίζει **δυναμικό (runtime) πολυμορφισμό**.

Παράδειγμα:

```
class Base {
public:
    virtual void sayHello() { cout << "Hello from Base\n"; }
};

class Derived : public Base {
public:
    void sayHello() override { cout << "Hello from Derived\n"; }
};

int main() {
    Base* ptr = new Derived();
    ptr->sayHello();  // Εκτυπώνει "Hello from Derived"
    delete ptr;
}
```

Χωρίς το `virtual`, θα εκτυπωνόταν **"Hello from Base"**, ακόμα κι αν το αντικείμενο ήταν τύπου `Derived`.

Λέξη-Κλειδί	Ρόλος
`public`	Ανοικτή πρόσβαση
`private`	Πρόσβαση μόνο εντός της ίδιας κλάσης
`protected`	Πρόσβαση μόνο από την ίδια και τις υποκλάσεις
`friend`	Ειδική πρόσβαση σε άλλες συναρτήσεις/κλάσεις
`virtual`	Υποστηρίζει δυναμικό binding (πολυμορφισμός)

1. **Δημιούργησε δύο βασικές κλάσεις `A` και `B`, καθεμία με μία μέθοδο `show()`, και μια κλάση `C` που κληρονομεί από τις δύο. Κάνε override τη `show()` στην `C`.**

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

class A {
public:
    void show() {
        cout << "Class A\n";
    }
};

class B {
public:
    void show() {
        cout << "Class B\n";
    }
};

class C : public A, public B {
public:
    void show() {
        cout << "Class C\n";
    }
};

int main() {
    C obj;
    obj.A::show();  // Κλήση από A
    obj.B::show();  // Κλήση από B
    obj.show();     // Κλήση από C
    return 0;
}
```

2. Και οι δύο βασικές κλάσεις έχουν ένα μέλος `int x`. Επίλυσε τη σύγκρουση στην κλάση `C`.

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

class A {
public:
    int x;
    A() { x = 10; }
};

class B {
public:
    int x;
    B() { x = 20; }
};

class C : public A, public B {
public:
    void printX() {
        cout << "A::x = " << A::x << endl;
        cout << "B::x = " << B::x << endl;
    }
};

int main() {
    C obj;
    obj.printX();
    return 0;
}
```

3. Δημιούργησε δύο κλάσεις `Person` και `Employee` που κληρονομούν εικονικά από `Identity`, ώστε να αποφευχθεί το diamond problem.

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

class Identity {
public:
    string name;
    Identity() { name = "Unnamed"; }
};

class Person : virtual public Identity {
public:
    void setPersonName(string n) { name = n; }
};

class Employee : virtual public Identity {
public:
    void setEmployeeName(string n) { name = n; }
};

class Manager : public Person, public Employee {
public:
    void display() {
        cout << "Manager Name: " << name << endl;
    }
};

int main() {
    Manager m;
    m.setPersonName("George");
    m.display();
    return 0;
}
```

4. Δημιούργησε κλάσεις `Flyable` και `Swimmable` με virtual μεθόδους, και μία κλάση `Duck` που τις υλοποιεί.

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

class Flyable {
public:
    virtual void fly() = 0;
};

class Swimmable {
public:
    virtual void swim() = 0;
};

class Duck : public Flyable, public Swimmable {
public:
    void fly() override {
        cout << "Duck is flying\n";
    }

    void swim() override {
        cout << "Duck is swimming\n";
    }
};

int main() {
    Duck d;
    d.fly();
    d.swim();
    return 0;
}
```

5. Επίδειξε τη σειρά κλήσεων constructors σε πολλαπλή κληρονομικότητα.

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

class A {
public:
    A() { cout << "Constructor A\n"; }
};

class B {
public:
    B() { cout << "Constructor B\n"; }
};

class C : public A, public B {
public:
    C() { cout << "Constructor C\n"; }
};

int main() {
    C obj;
    return 0;
}
```