## Topic 1: Case 1

```cpp
#include <iostream>
using namespace std;
class Father
{
private:
    int money;
protected:
    int gold;
public:
    int land;
public:
    Father()
    {
        money = 1;
        gold = 2;
        land = 3;
    }
};
class Son : public Father
{
public:
    int get_prot()
    {
        return gold;
    }
};

class GrandSon1 : public Son
{
};
class GrandSon2 : protected Son
{
public:
    int get_prot()
    {
        return gold;
    }
    int get_prot2()
    {
        return land;
    }
};
class GrandSon3 : private Son
{
public:
    int get_private()
    {
        return gold;
    }
    int get_private2()
    {
        return land;
    }
};
int main()
{
    Son test2;
    GrandSon1 test3;
    GrandSon2 test4;
    GrandSon3 test5;
    cout << "Son Public Class from Father members" << endl;
    cout << "Private not accessible from Father" << endl;
    cout << "Protected = " << test2.get_prot() << endl;
    cout << "Public = " << test2.land << endl;
    cout << "GrandSon1 Public Class from Son members" << endl;
    cout << "Private not accessible from Father" << endl;
    cout << "Protected = " << test3.get_prot() << endl;
    cout << "Public = " << test3.land << endl;
    cout << "GrandSon2 Protected Class from Son members" << endl;
    cout << "Private not accessible from Father" << endl;
    cout << "Protected = " << test4.get_prot() << endl;
    cout << "Protected = " << test4.get_prot2() << endl;
    cout << "GrandSon3 Private Class from Son members" << endl;
    cout << "Private not accessible from Father" << endl;
    cout << "Private = " << test5.get_private() << endl;
    cout << "Private = " << test5.get_private2() << endl;
}
```

Topic 1: Case 2

```cpp
#include <iostream>
using namespace std;
class Father
{
private:
    int money;
protected:
    int gold;
public:
    int land;
public:
    Father()
    {
        money = 1;
        gold = 2;
        land = 3;
    }
};
class Son : protected Father
{
public:
    int get_prot()
    {
        return gold;
    }
    int get_prot2()
    {
        return land;
    }
};
class GrandSon1 : public Son
{
public:
    int get_prot()
    {
        return gold;
    }
    int get_prot2()
    {
        return land;
    }
};
class GrandSon2 : protected Son
{
public:
    int get_prot()
    {
        return gold;
    }
    int get_prot2()
    {
        return land;
    }
};
class GrandSon3 : private Son
{
public:
    int get_private()
    {
        return gold;
    }
    int get_private2()
    {
        return land;
    }
};

int main()
{
    Son test2;
    GrandSon1 test3;
    GrandSon2 test4;
    GrandSon3 test5;
    cout << "Son Protected Class from Father members" << endl;
    cout << "Private not accessible from Father" << endl;
    cout << "Protected = " << test2.get_prot() << endl;
    cout << "Protected = " << test2.get_prot2() << endl;
    cout << "GrandSon1 Public Class from Son members" << endl;
    cout << "Private not accessible from Father" << endl;
    cout << "Protected = " << test3.get_prot() << endl;
    cout << "Protected = " << test3.get_prot2() << endl;
    cout << "GrandSon2 Protected Class from Son members" << endl;
    cout << "Private not accessible from Father" << endl;
    cout << "Protected = " << test4.get_prot() << endl;
    cout << "Protected = " << test4.get_prot2() << endl;
    cout << "GrandSon3 Private Class from Son members" << endl;
    cout << "Private not accessible from Father" << endl;
    cout << "Private = " << test5.get_private() << endl;
    cout << "Private = " << test5.get_private2() << endl;
}
```

## Topic 1: Case 3

```
#include <iostream>
using namespace std;
class Father
{
private:
    int money;
protected:
    int gold;
public:
    int land
public:
    Father()
    {
        money = 1;
        gold = 2;
        land = 3;
    }
};
class Son : private Father
{
public:
    int get_private()
    {
        return gold;
    }
    int get_private2()
    {
        return land;
    }
};
class GrandSon1 : public Son
{
public:
};
class GrandSon2 : protected Son
{
public:
};
class GrandSon3 : private Son
{
public:
};
int main()
{
    Son test2;
    GrandSon1 test3;
    GrandSon2 test4;
    GrandSon3 test5;
    cout << "Son Private Class from Father members" << endl;
    cout << "Private not accessible from FAther" << endl;
    cout << "Private = " << test2.get_private() << endl;
    cout << "Private = " << test2.get_private2() << endl;
    cout << "GrandSon1 Public Class from Son members" << endl;
    cout << "Private not accessible from Son" << endl;
    cout << "GrandSon2 Protected Class from Son members" << endl;
    cout << "Private not accessible from Son" << endl;
    cout << "GrandSon3 Private Class from Son members" << endl;
    cout << "Private not accessible from Son" << endl;
}
```

| Class | | In Son Class | | | In Grandson Class | | |
|---|---|---|---|---|---|---|---|
| Son | Grandson | Money | Gold | Land | Money | Gold | Land |
| Public | Public | No | Yes | Yes | No | Yes | Yes |
| Protected | Public | No | Yes | Yes | No | Yes | Yes |
| Private | Public | No | Yes | Yes | No | No | No |
| Public | Protected | No | Yes | Yes | No | Yes | Yes |
| Protected | Protected | No | Yes | Yes | No | Yes | Yes |
| Private | Protected | No | Yes | Yes | No | No | No |
| Public | Private | No | Yes | Yes | No | Yes | Yes |
| Protected | Private | No | Yes | Yes | No | Yes | Yes |
| Private | Private | No | Yes | Yes | No | No | No |

## Topic 2: Single

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

class A
{
private:
    int x = 1;

protected:
    int y = 2;

public:
    int z = 3;
};

class B : public A
{
public:
    int gety() { return y; }
};

int main()
{
    B test;
    cout << "Private x not accessible out of base class" << endl;
    cout << "protected y = " << test.gety() << endl;
    cout << "public z = " << test.z << endl;
}
```

## Topic 2: Multi-Level

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

class A
{
private:
    int x = 1;

protected:
    int y = 2;

public:
    int z = 3;
};
class B : public A
{
};
class C : public B
{
public:
    int gety() { return y; }
};

int main()
{
    C test;
    cout << "Private x not accessible out of base class" << endl;
    cout << "protected y = " << test.gety() << endl;
    cout << "public z = " << test.z << endl;
}
```

## Topic 2: Multiple

```cpp
// MU
#include <iostream>
using namespace std;
class A
{
private:
    int x = 1;
protected:
    int y = 2;
public:
    int z = 3;
};
class B
{
private:
    int p = 4;
protected:
    int q = 5;
public:
    int r = 6;
};
class C : public A, public B
{
public:
    int gety() { return y; }
    int getq() { return q; }
};
int main()
{
    C test;
    cout << "Private x, p not accessible out of base class" << endl;
    cout << "protected y = " << test.gety() << endl;
    cout << "public z = " << test.z << endl;
    cout << "protected q = " << test.getq() << endl;
    cout << "public r = " << test.r << endl;
}
```

## Topic 2: Hierarchical

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

class A
{
private:
    int x = 1;

protected:
    int y = 2;

public:
    int z = 3;
};
class B : public A
{
public:
    int gety() { return y; }
};
class C : public A
{
public:
    int gety() { return y; }
};

int main()
{
    B b;
    C test;
    cout << "Private x not accessible out of base class" << endl;
    cout << "protected y = " << b.gety() << endl;
    cout << "public z = " << b.z << endl;
    cout << "protected y = " << test.gety() << endl;
    cout << "public z = " << test.z << endl;
}
```

## Topic 2: Hybrid

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

class A
{
private:
    int x = 1;

protected:
    int y = 2;

public:
    int z = 3;
};
class B : virtual public A
{
};
class C : virtual public A
{
};
class D : public B, public C
{
public:
    int gety()
    {
        return y;
    }
};

int main()
{
    D test;
    cout << "protected y = " << test.gety() << endl;
    cout << "public z =" << test.z << endl;
}
```

## Topic 3: Single

```cpp
#include <iostream>
using namespace std;
class A
{
private:
    int ax;
public:
    A()
    {
        ax = 12;
    }
    int getax() { return ax; }
    ~A()
    {
        cout << "A Destructor is Called" << endl;
    }
};
class B : public A
{
private:
    int bx;
public:
    B()
    {
        bx = 34;
    }
    int sum()
    {
        return getax() + bx;
    }
    ~B()
    {
        cout << "B Destructor is Called" << endl;
    }
};
int main()
{
    B b;
    cout << "The sum is = " << b.sum() << endl;
}
```

## Topic 3: Multi-Level

```
#include <iostream>
using namespace std;
class A
{
private:
    int ax;
public:
    A()
    {
        ax = 12;
    }
    int getax() { return ax; }
    ~A()
    {
        cout << "A Destructor is Called" << endl;
    }
};
class B : public A
{
private:
    int bx;
public:
    B()
    {
        bx = 34;
    }
    int getbx() { return bx; }
    ~B()
    {
        cout << "B Destructor is Called" << endl;
    }
};
class C : public B
{
private:
    int cx;
public:
    C()
    {
        cx = 56;
    }
    int sum()
    {
        return getax() + getbx() + cx;
    }
    ~C()
    {
        cout << "C Destructor is Called" << endl;
    }
};
int main()
{
    C c;
    cout << "The sum is = " << c.sum() << endl;
}
```

## Topic 3: Multiple

```
#include <iostream>
using namespace std;
class A
{
private:
    int ax;

public:
    A()
    {
        ax = 12;
    }
    int getax() { return ax; }
    ~A()
    {
        cout << "A Destructor is Called" << endl;
    }
};
class B
{
private:
    int bx;

public:
    B()
    {
        bx = 34;
    }
    int getbx() { return bx; }
    ~B()
    {
        cout << "B Destructor is Called" << endl;
    }
};
class C : public A, public B
{
private:
    int cx;

public:
    C()
```

```cpp
    {
        cx = 56;
    }
    int sum()
    {
        return getax() + getbx() + cx;
    }
    ~C()
    {
        cout << "C Destructor is Called" << endl;
    }
};
int main()
{
    C c;
    cout << "The sum is = " << c.sum() << endl;
}
```

## Topic 3: Hierarchical

```cpp
#include <iostream>
using namespace std;
class A
{
private:
    int ax;

public:
    A()
    {
        ax = 12;
    }
    int getax() { return ax; }
    ~A()
    {
        cout << "A Destructor is Called" << endl;
    }
};
class B : public A
{
private:
    int bx;

public:
    B()
    {
        bx = 34;
    }
    int getbx() { return bx; }
    ~B()
    {
        cout << "B Destructor is Called" << endl;
    }
};
class C : public A
{
private:
    int cx;

public:
    C()
    {
        cx = 56;
    }
    int sum(B &b)
    {
        return getax() + b.getbx() + cx; // getbx() doesn't work why??
    }
    ~C()
    {
        cout << "C Destructor is Called" << endl;
    }
};
int main()
{
    B test;
    C c;
    cout << "The sum is = " << c.sum(test) << endl;
}
```

**Topic 3: Hybrid**

```
#include <iostream>
using namespace std;
class A
{
private:
    int ax;

public:
    A()
    {
        ax = 12;
    }
    int getax() { return ax; }
    ~A()
    {
        cout << "A Destructor is Called" << endl;
    }
};
class B : virtual public A
{
```

```
private:
    int bx;

public:
    B()
    {
        bx = 34;
    }
    int getbx() { return bx; }
    ~B()
    {
        cout << "B Destructor is Called" << endl;
    }
};
class C : virtual public A
{
private:
    int cx;

public:
    C()
    {
        cx = 56;
    }
    int getcx()
    {
        return cx;
    }
    ~C()
    {
        cout << "C Destructor is Called" << endl;
    }
};
class D : public B, public C
{
private:
    int dx;

public:
    D()
    {
        dx = 78;
    }
    int sum()
    {
        return getax() + getbx() + getcx() + dx;
    }
    ~D()
    {
        cout << "D Destructor is Called" << endl;
    }
};

int main()
{
    D d;
    cout << "The sum is = " << d.sum() << endl;
}
```