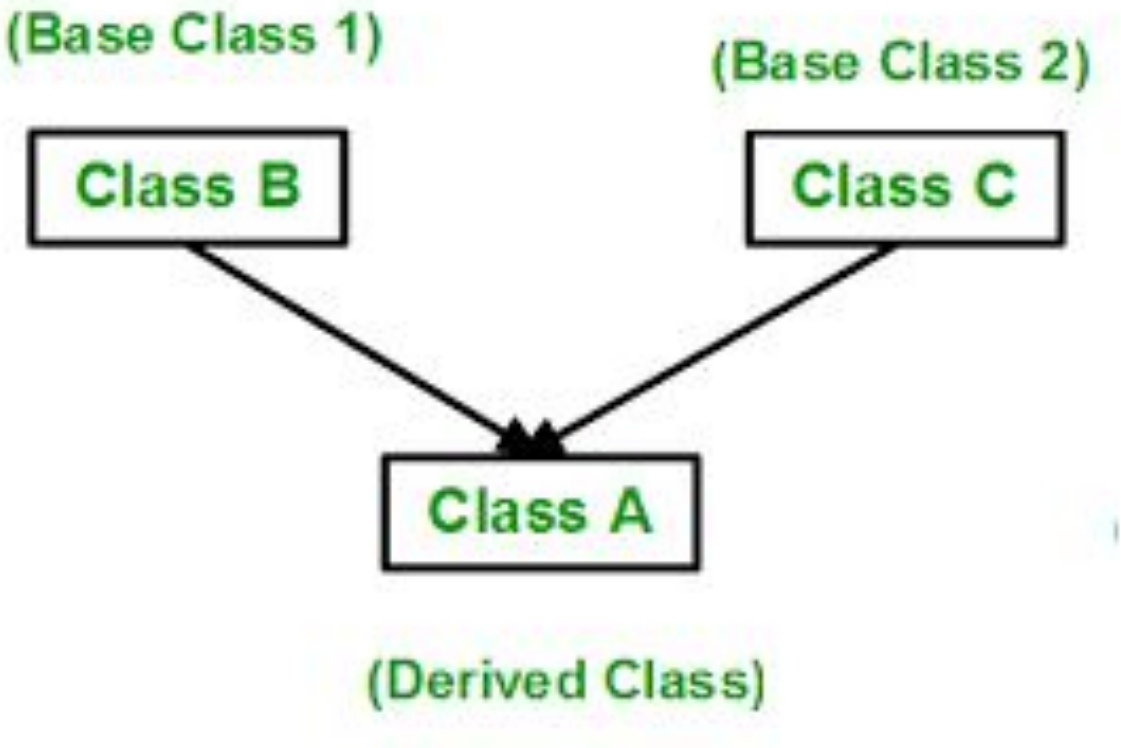
Slide Set 3

Inheritance: Derived Class declaration, Public, Private and Protected Inheritance, friend function and Inheritance, Overriding member function, Forms of inheritance, virtual base class, Abstract class, Constructor and Inheritance, Destructor and Inheritance, Advantage and disadvantage of Inheritance.

Inheritance - one class acquires the members of another class



Derived Class declaration

```
class A : public B, public C {
...
};
```

// we use public most of the time (explained ahead)

```
class Base {
protected: int i;
public: Base(int a){i=a;}
void baseShow() {cout<<"i="<<i<<endl;}
};
class Child : public Base {
int j;
public: Child(int a, int b) : Base(a) {j=b;}
void childShow(){cout<<"i="<<i<<",j="<<j<<endl;}
};
int main(void) {
Base b(55); b.baseShow();
Child c(10,20); c.childShow();
}
```

Output

i=55

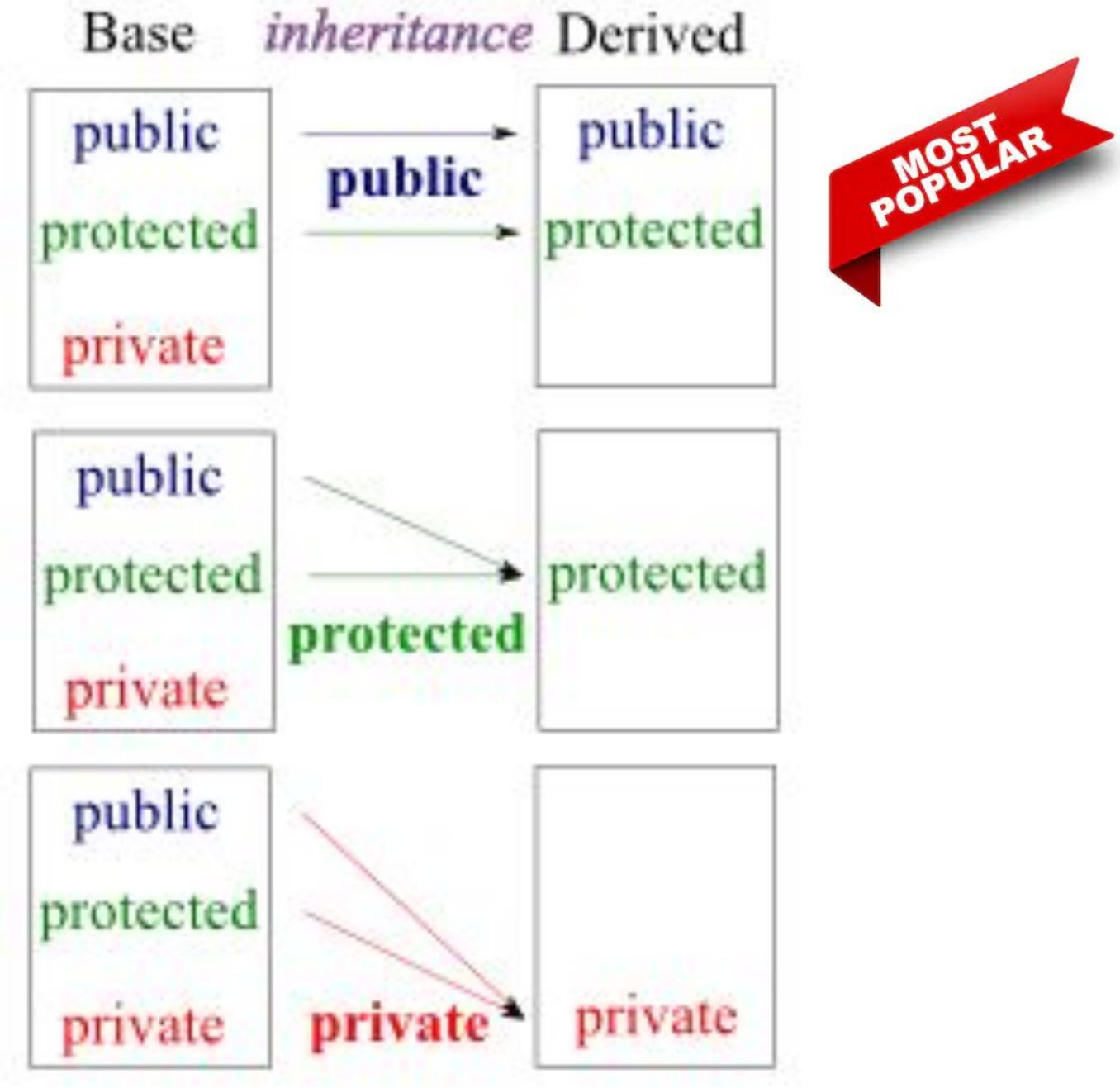
i=10,j=20

Access specifier

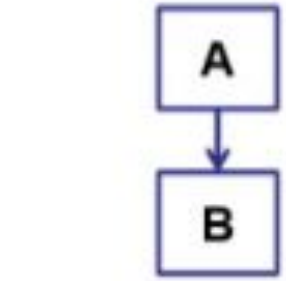
Access	public	**protected**	private
Same class	✔	✔	✔
Derived classes	✔	✔	❌
Outside classes	✔	❌	❌

Three modes of inheritance

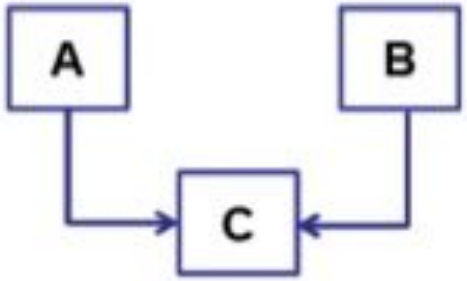
1. **Public**
2. Protected
3. private


Base
inheritance
Derived
public
protected
private
public
public
protected
public
protected
private
protected
protected
public
protected
private
private
private
MOST POPULAR

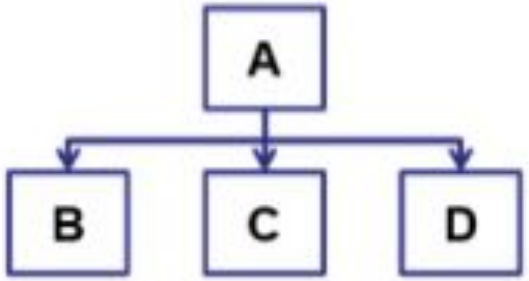
Types of inheritances in C++



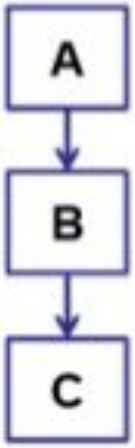
Single Inheritance



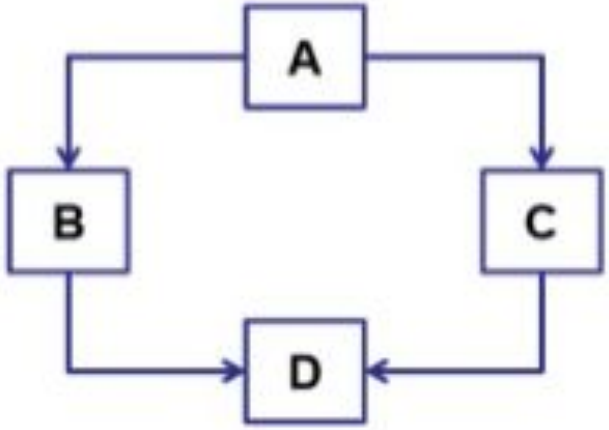
Multiple Inheritance



Hierarchical Inheritance



Multilevel Inheritance



Hybrid Inheritance

Friend function and Inheritance

- Friendship is not inherited
- What is the output of the following program?

```
class A {
protected: int i;
public: A(int a) { i = a;}
friend void show();
};
class B: public A {
int j;
public: B(int a, int b) : A(a) {j=b;}
};
void show() {
B b(-2,55);
cout<<b.i<<", "<<b.j; }

int main() {show();}
```

Output

[Error] 'int B::j' is private

So parent’s friend is not child’s friend,
what about vice versa?

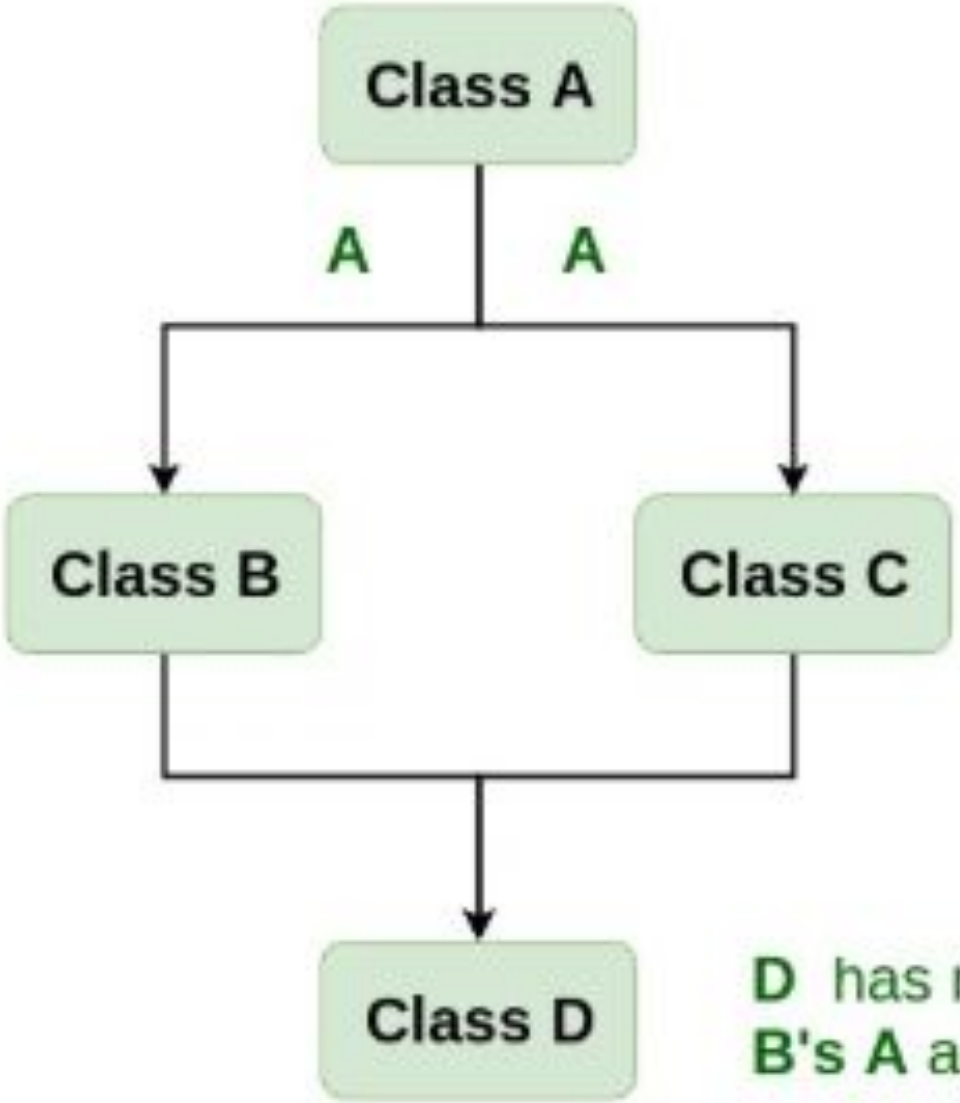
Overriding member function

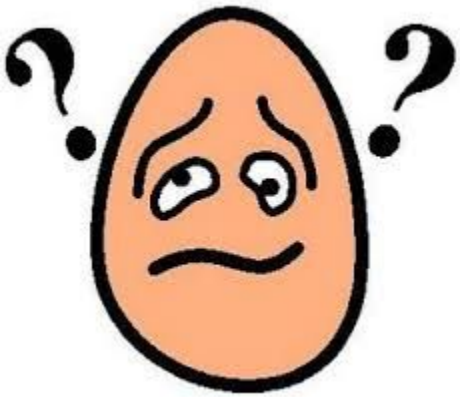
- It is like creating a *new version* of an old function, in the child class.

```
class Base {
public: void disp(){cout<<"Base"<<endl; }
};
class Derived: public Base{
public: void disp(){cout<<"Derived"<<endl; }
};
int main() {
Derived d;
d.disp(); }
```

Virtual base class

- They are used to prevent the confusion or duplicity among child classes during inheritance.
- Consider the following situation:





D has now 2 **A**:
B's A and **C's A**

```
class A {
public: void show() {cout<<"In A"<<endl;}
};

class B : public A {};
class C : public A {};
class D : public B, public C {};

int main() {D d; d.show(); }
```

Output

[Error] request for member 'show' is ambiguous

Need ***virtual*** keyword to fix this error

```
class A {
public: void show() {cout<<"In A"<<endl;}
};

class B : public virtual A {};
class C : public virtual A {};
class D : public B, public C {};

int main() {D d; d.show(); }
```

Abstract base class - interface

- It describes the capabilities of a C++ class without committing to a particular implementation
- A class is made abstract by declaring at least one of its functions as **pure virtual** function i.e. by placing **"= 0"** in its declaration

```
class Animal{
public:
virtual void sound() = 0;
void sleeping() {cout<<"Sleeping"; }
};
class Dog: public Animal{
public:
void sound() {cout<<"Woof"<<endl;}
};

int main(){
   Dog obj;  obj.sound();  obj.sleeping();
}
```

OUTPUT:

WOOF

SLEEPING

```
class Animal{
public:
virtual void sound() = 0;
void sleeping() {cout<<"Sleeping"; }
};
class Dog: public Animal{
public:
void sound() {cout<<"Woof"<<endl;}
};

int main(){
Animal *obj = new Dog;
obj->sound();
}
```

OUTPUT:

WOOF

Constructor order

- Base class constructors are always called first in the derived class constructors.
- For multilevel inheritance, constructors are called in the order of inheritance

Method of inheritance

```
class B:public A {
};

class A:public B, public C
{
};
```

Order of execution

A(); base constructor

B();derived constructor

B(); base(first)

C(); base(second)

A(); derived

Destructors order in inheritance

- Destructors are called in ***reverse*** order as that of constructors

  class A:public B, public C {…};

  Order of execution of constructors and destructors:

  ~A();

  ~C();

  ~B();

Pros and cons of inheritance

1. Reuse the methods and data of a class
2. Extend the existing class by adding new data and functions
3. Modify the existing class by overloading its methods
4. Adds extra memory overload for the compiler as it has to keep records of the parent as well as the child class