Polymorphism

Early and Late binding
virtual functions

Early binding

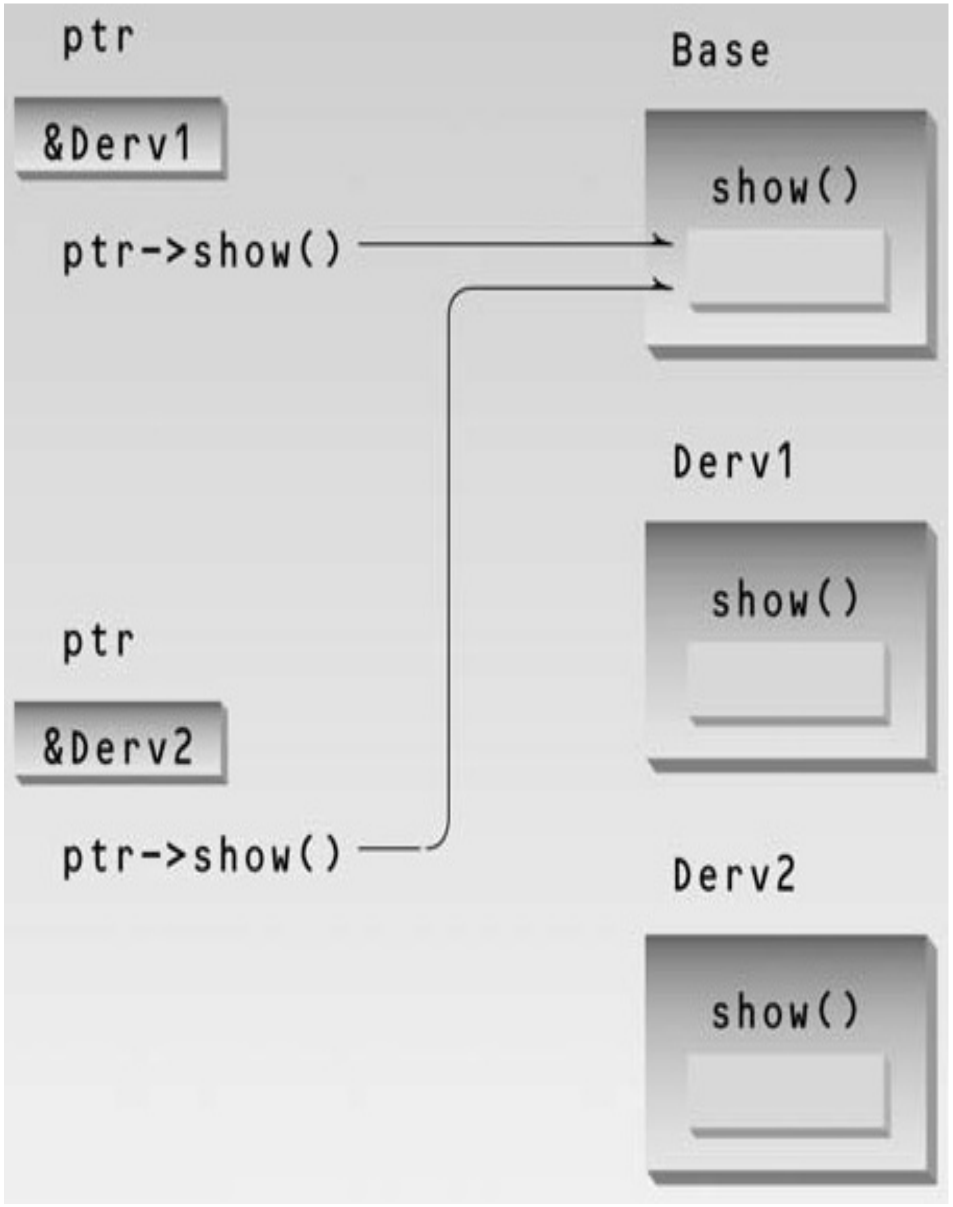
```
class Base //base class
{
public:
void show( ) //normal function
    { cout << “Base\n”; }
};
class Derv1 : public Base //derived
class 1
{
public:
void show( )
  { cout << “Derv1\n”; }
};
class Derv2 : public Base //derived
class 2
{
public:
void show( )
   { cout << “Derv2\n”; }
};
```

```
int main()
{
Base b;
Derv1 dv1; //object of derived class 1
Derv2 dv2; //object of derived class 2
Base* ptr; //pointer to base class
ptr = &b;
prt->show( );
ptr = &dv1; //put address of dv1 in
pointer
ptr->show( ); //execute show()
ptr = &dv2; //put address of dv2 in
pointer
ptr->show( ); //execute show()
return 0;
}
```

**OUTPU
T**

```
Base
Base
Base
```

Early binding



```
int main()
{
Base b;
Derv1 dv1; //object of derived class 1
Derv2 dv2; //object of derived class 2
Base* ptr; //pointer to base class
ptr = &b;
prt->show( );
ptr = &dv1; //put address of dv1 in
pointer
ptr->show( ); //execute show()
ptr = &dv2; //put address of dv2 in
pointer
ptr->show( ); //execute show()
return 0;
}
```

OUTPUT

```
Base
Base
Base
```

Late binding virtual member functions

```
class Base //base class
{
public:
virtual void show( ) //virtual function
{ cout << “Base\n”; }
};
class Derv1 : public Base //derived
class 1
{public:
void show( )
{ cout << “Derv1\n”; }
};
class Derv2 : public Base //derived
class 2
{
public:
void show( )
{ cout << “Derv2\n”; }
};
```

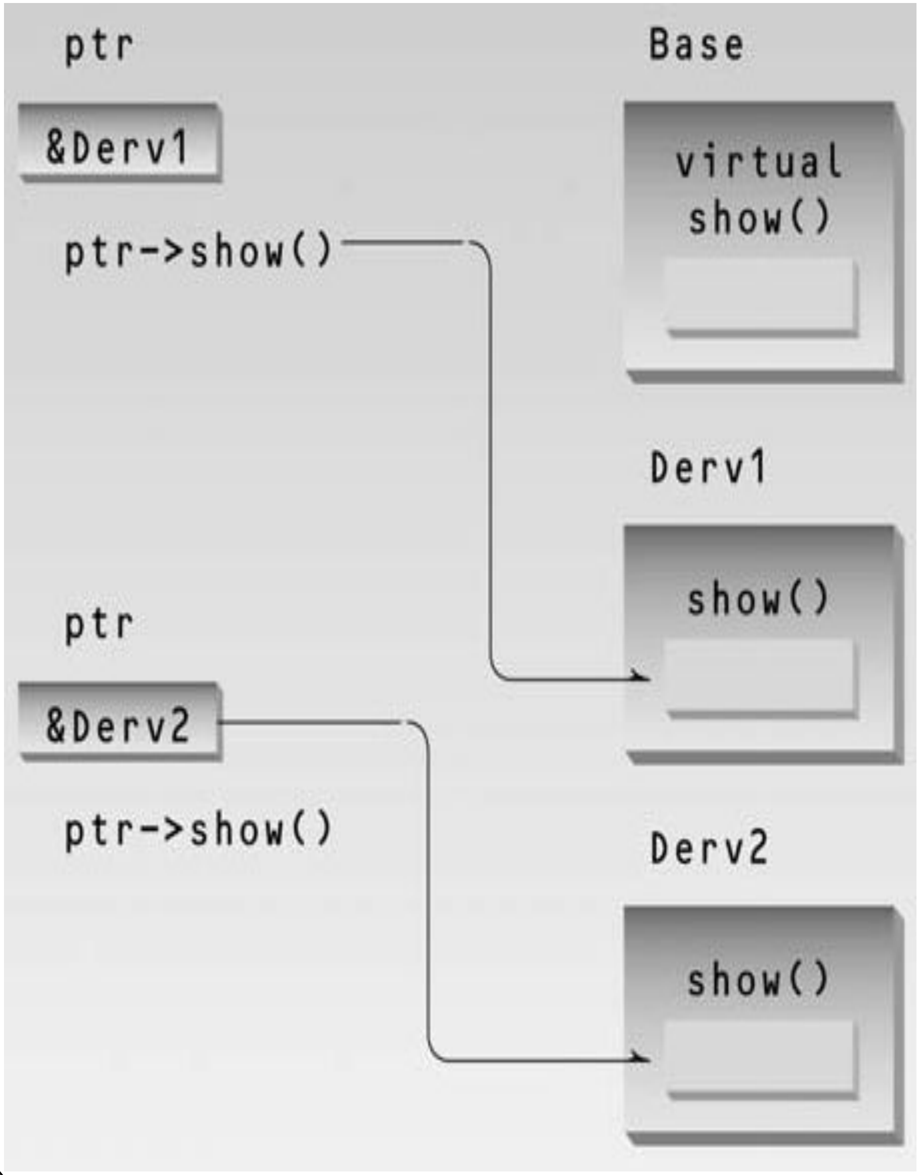
KEYWORD

```
int main()
{
Base b;
Derv1 dv1; //object of derived class 1
Derv2 dv2; //object of derived class 2
Base* ptr; //pointer to base class
ptr = &b;
prt->show( );
ptr = &dv1; //put address of dv1 in
pointer
ptr->show( ); //execute show()
ptr = &dv2; //put address of dv2 in
pointer
ptr->show( ); //execute show()
return 0;
}
```

OUTPUT

```
Base
Derv1
Derv2
```

Late binding
virtual member functions



```
int main()
{
Base b;
Derv1 dv1; //object of derived class 1
Derv2 dv2; //object of derived class 2
Base* ptr; //pointer to base class
ptr = &b;
prt->show( );
ptr = &dv1; //put address of dv1 in
pointer
ptr->show( ); //execute show()
ptr = &dv2; //put address of dv2 in
pointer
ptr->show( ); //execute show()
return 0;
}
```

OUTPUT

```
Base
Derv1
Derv2
```

Abstract Classes and Pure Virtual Functions

```
class Base //base class
{
public:
  virtual void show( ) = 0; //pure virtual
function
};
class Derv1 : public Base //derived class 1
{
public:
  void show( )
      { cout << “Derv1\n”; }
};
class Derv2 : public Base //derived class 2
{
public:
  void show( )
      { cout << “Derv2\n”; }
```

Abstract Classes and Pure Virtual Functions

```
int main( )
{
// Base bad;  //can't make object from abstract
class
Base* arr[2];     //array of pointers to base class
Derv1 dv1;   //object of derived class 1
Derv2 dv2;   //object of derived class 2
arr[0] = &dv1; //put address of dv1 in array
arr[1] = &dv2; //put address of dv2 in array
arr[0]->show( ); //execute show() in both objects
arr[1]->show( );
return 0;
}
```

OUTPUT

```
Derv1
Derv2
```