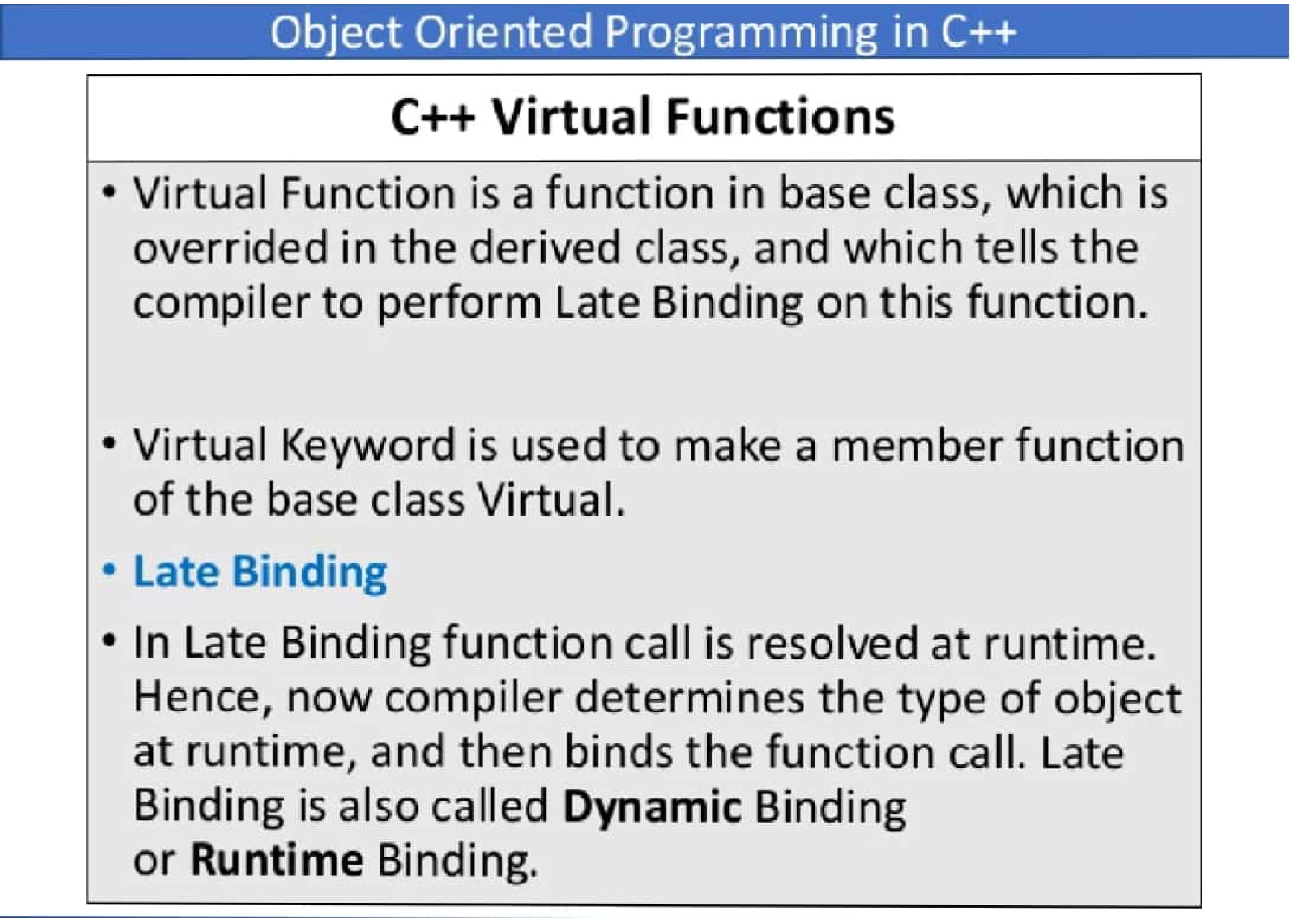
Object Oriented Programming in C++

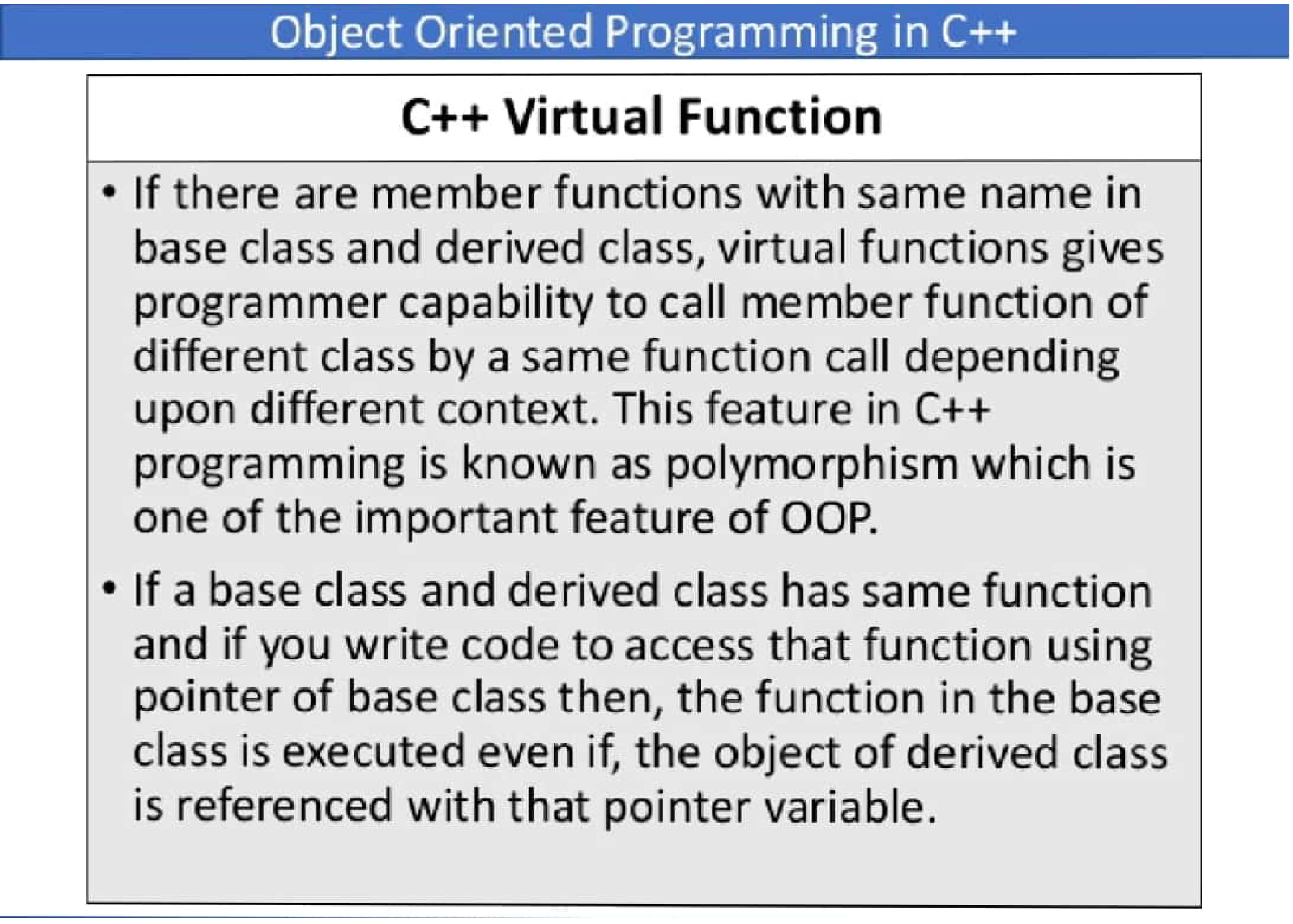
Virtual Functions in C++



Object Oriented Programming in C++

C++ Virtual Functions

- Virtual Function is a function in base class, which is overrided in the derived class, and which tells the compiler to perform Late Binding on this function.
- Virtual Keyword is used to make a member function of the base class Virtual.
- **Late Binding**
- In Late Binding function call is resolved at runtime. Hence, now compiler determines the type of object at runtime, and then binds the function call. Late Binding is also called **Dynamic** Binding or **Runtime** Binding.



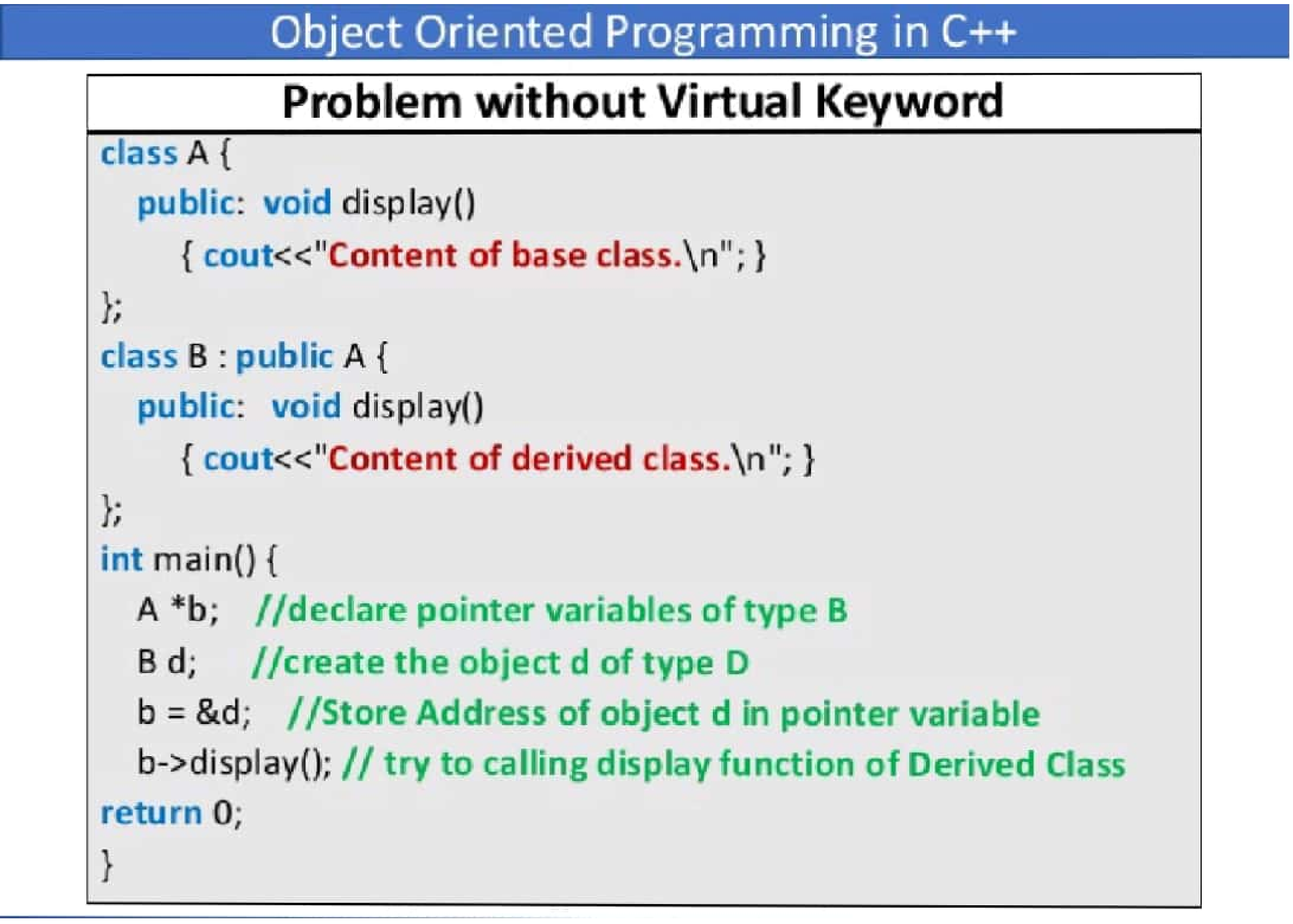
C++ Virtual Function

- If there are member functions with same name in base class and derived class, virtual functions gives programmer capability to call member function of different class by a same function call depending upon different context. This feature in C++ programming is known as polymorphism which is one of the important feature of OOP.
- If a base class and derived class has same function and if you write code to access that function using pointer of base class then, the function in the base class is executed even if, the object of derived class is referenced with that pointer variable.

C++ Virtual Function

- A **virtual function** is a member function of class that is declared within a base class and re-defined in derived class.
- When you want to use same function name in both the base and derived class, then the function in base class is declared as virtual by using the **virtual** keyword and again re-defined this function in derived class without using virtual keyword.
- **Syntax**

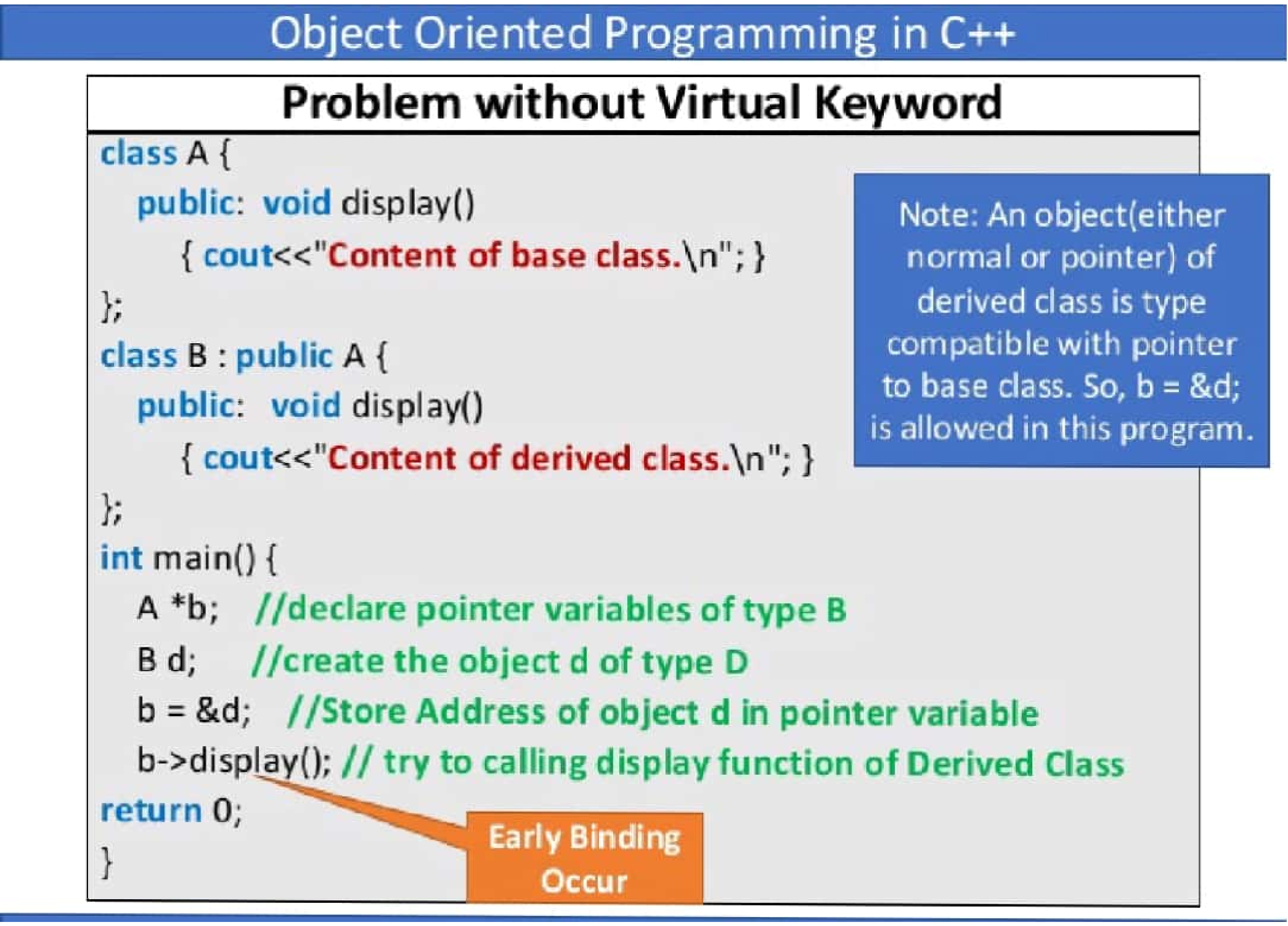
```
virtual return_type function_name()
{
 .......
 .......
}
```



Object Oriented Programming in C++

Problem without Virtual Keyword

```
class A {
   public:  void display()
       { cout<<"Content of base class.\n"; }
};
class B : public A {
   public:   void display()
       { cout<<"Content of derived class.\n"; }
};
int main() {
   A *b;    //declare pointer variables of type B
   B d;     //create the object d of type D
   b = &d;    //Store Address of object d in pointer variable
   b->display(); // try to calling display function of Derived Class
return 0;
}
```



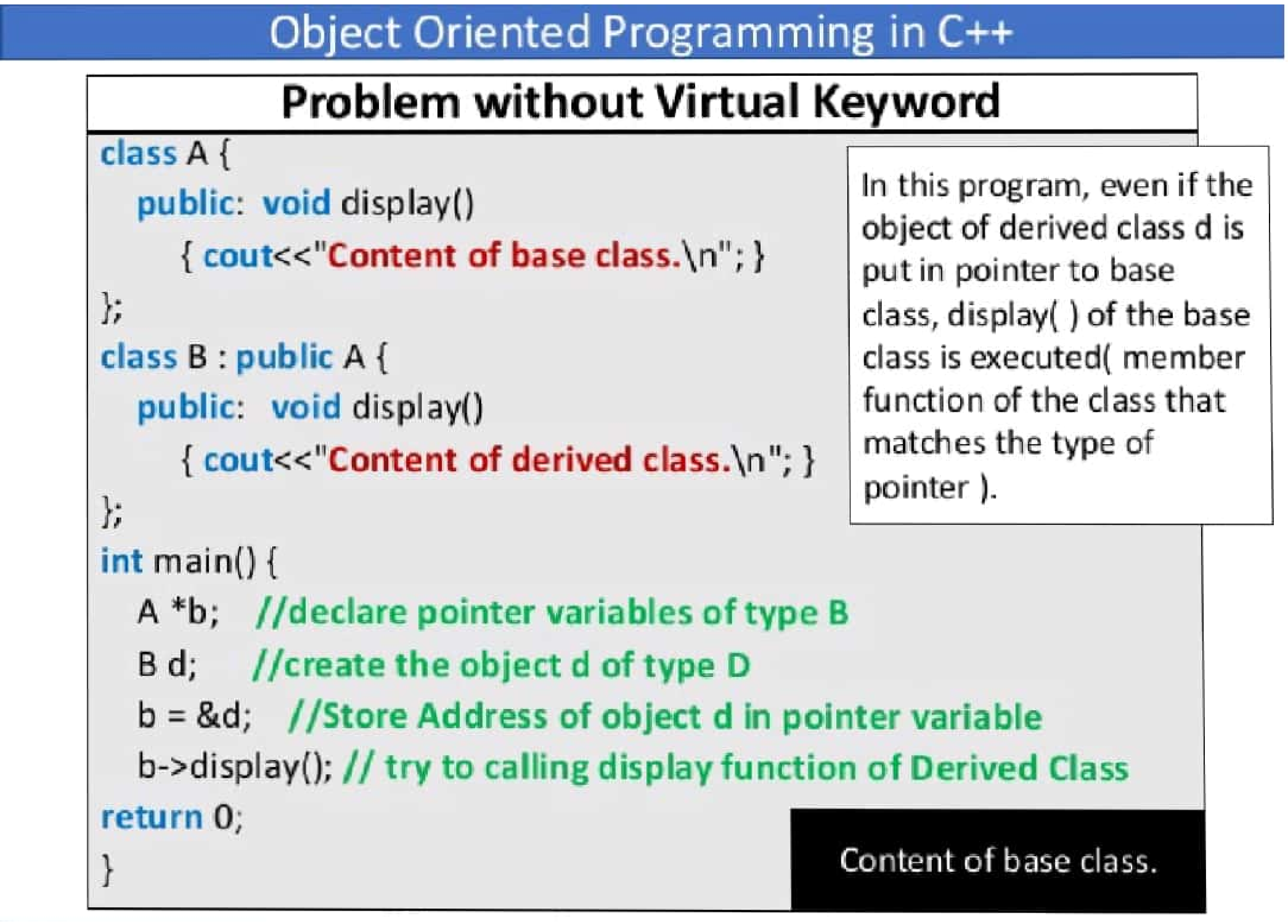
Object Oriented Programming in C++

Problem without Virtual Keyword

```
class A {
   public:  void display()
       { cout<<"Content of base class.\n"; }
};
class B : public A {
   public:   void display()
       { cout<<"Content of derived class.\n"; }
};
int main() {
   A *b;    //declare pointer variables of type B
   B d;     //create the object d of type D
   b = &d;   //Store Address of object d in pointer variable
   b->display(); // try to calling display function of Derived Class
return 0;
}
```

Note: An object(either normal or pointer) of derived class is type compatible with pointer to base class. So, b = &d; is allowed in this program.

Early Binding Occur



Object Oriented Programming in C++

Problem without Virtual Keyword

```
class A {
   public:  void display()
       { cout<<"Content of base class.\n"; }
};
class B : public A {
   public:   void display()
       { cout<<"Content of derived class.\n"; }
};
int main() {
   A *b;    //declare pointer variables of type B
   B d;     //create the object d of type D
   b = &d;    //Store Address of object d in pointer variable
   b->display(); // try to calling display function of Derived Class
return 0;
}
```

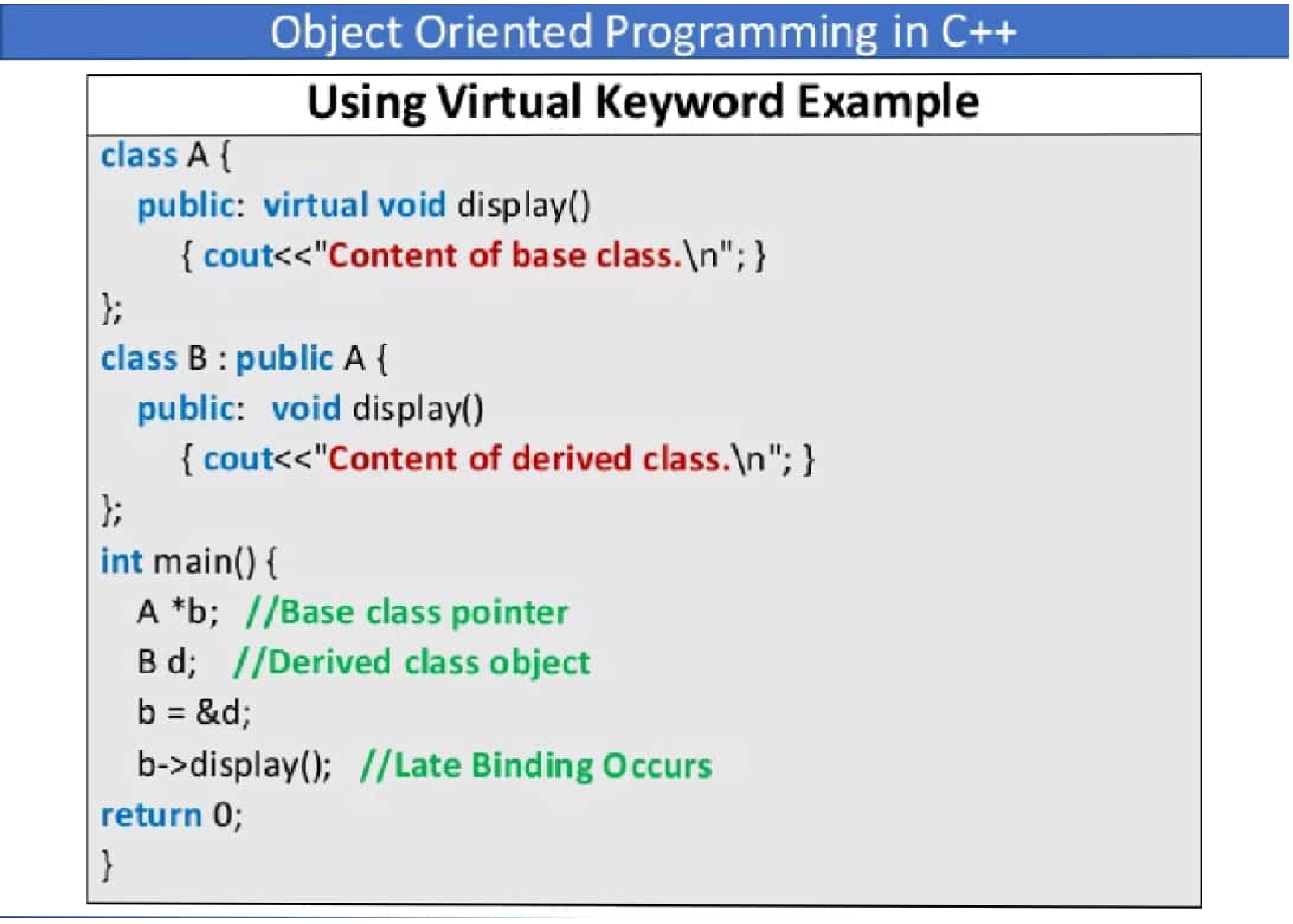
In this program, even if the object of derived class d is put in pointer to base class, display( ) of the base class is executed(member function of the class that matches the type of pointer).

```
Content of base class.
```

Scanned by CamScanner

Virtual Functions

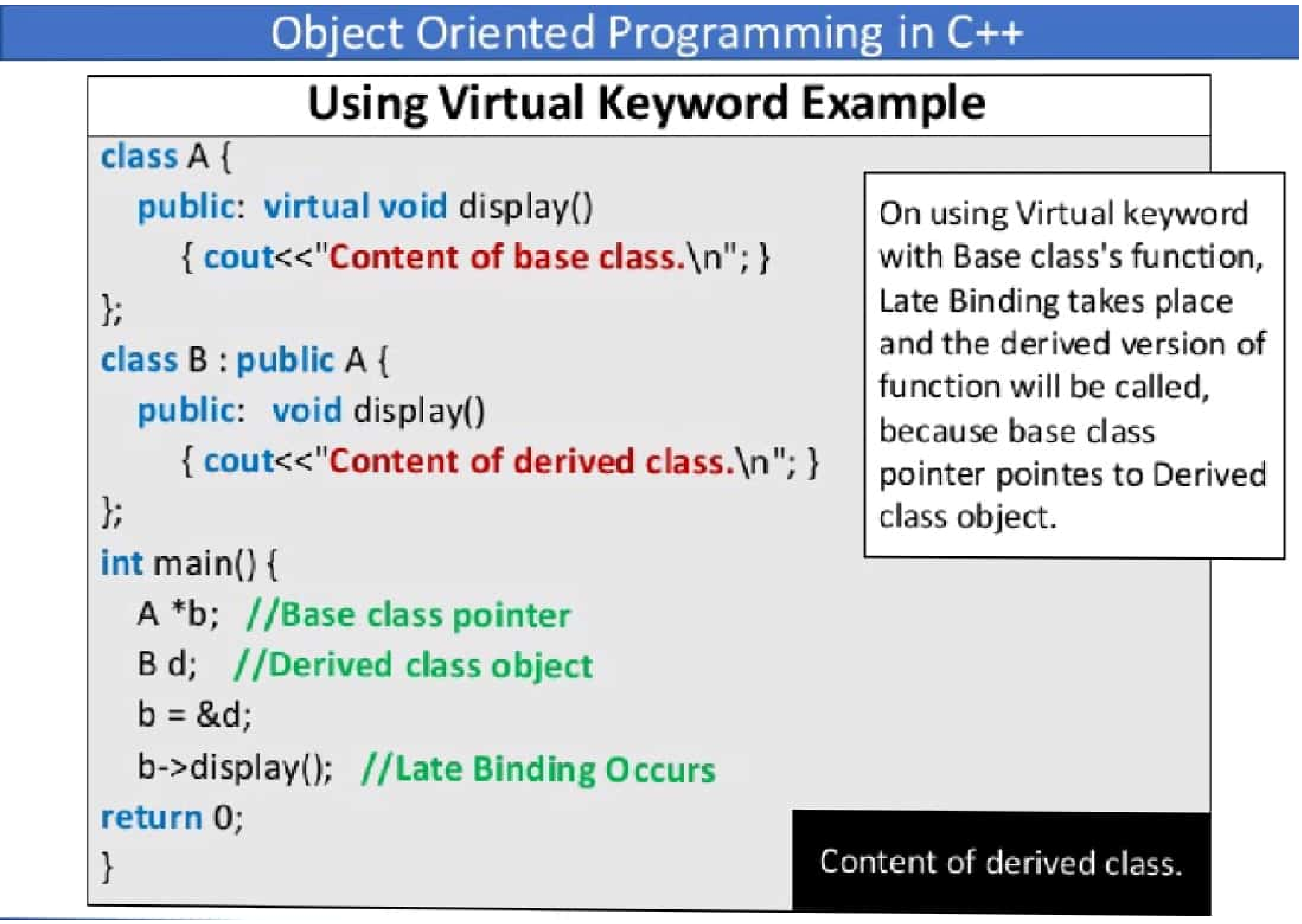
- If we want to execute the member function of derived class then, you can declare display( ) in the base class virtual which makes that function existing in appearance only but, you can't call that function. In order to make a function virtual, you have to add keyword virtual in front of a function.
- Let see an example in next slide.



Object Oriented Programming in C++

Using Virtual Keyword Example

```
class A {
   public:  virtual void display()
      { cout<<"Content of base class.\n"; }
};
class B : public A {
   public:   void display()
      { cout<<"Content of derived class.\n"; }
};
int main() {
   A *b;   //Base class pointer
   B d;   //Derived class object
   b = &d;
   b->display();   //Late Binding Occurs
return 0;
}
```



Object Oriented Programming in C++

Using Virtual Keyword Example

```
class A {
   public:  virtual void display()
       { cout<<"Content of base class.\n"; }
};
class B : public A {
   public:   void display()
       { cout<<"Content of derived class.\n"; }
};
int main() {
   A *b;   //Base class pointer
   B d;    //Derived class object
   b = &d;
   b->display();   //Late Binding Occurs
return 0;
}
```

On using Virtual keyword with Base class's function, Late Binding takes place and the derived version of function will be called, because base class pointer pointes to Derived class object.

```
Content of derived class.
```

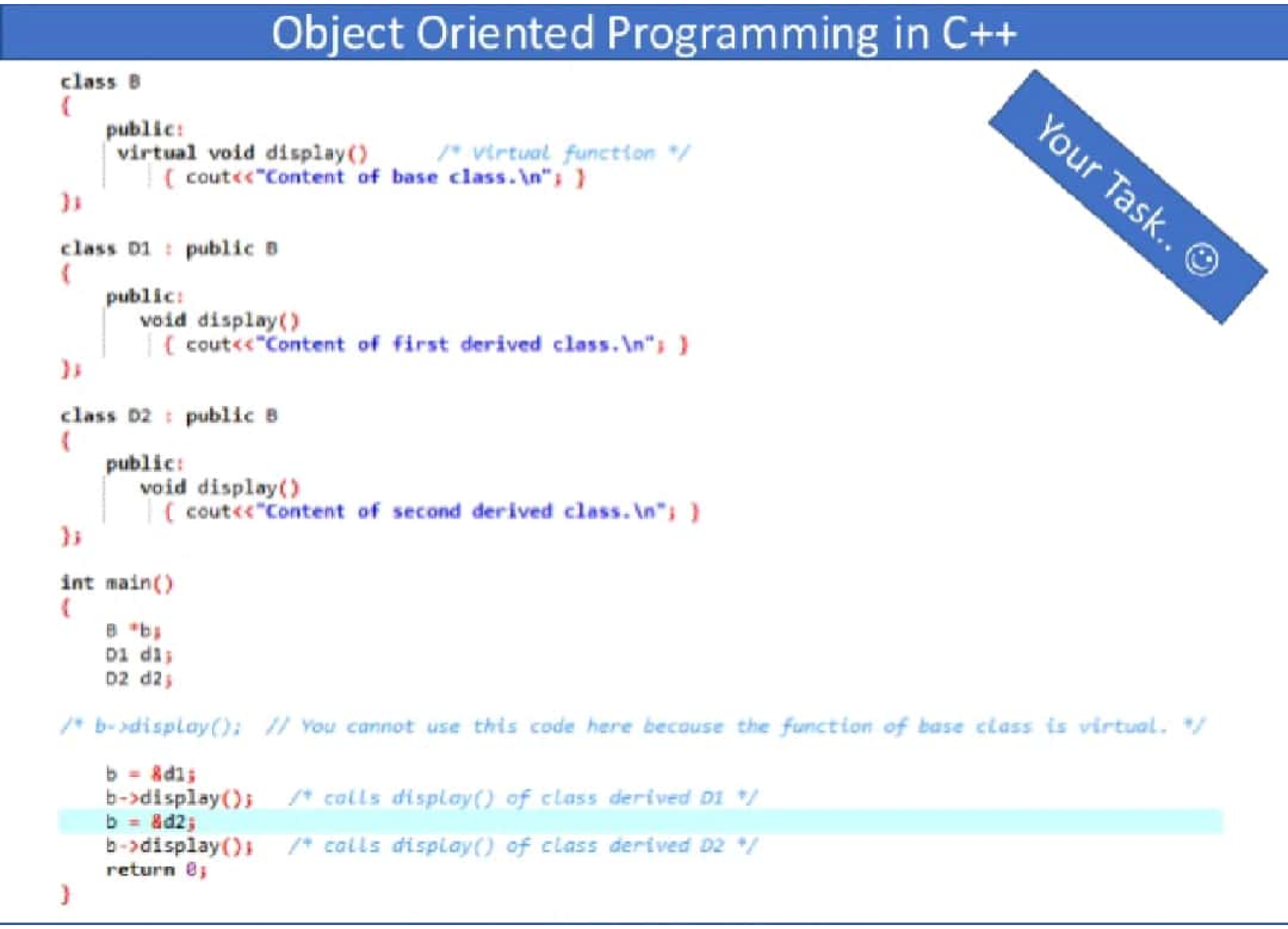
One More Example of Virtual Keyword-1

```
class A
{
        public:
        virtual void show() {
                cout<<"Hello base class\n";
        }
};
class B : public A {
        public:
        void show() {
                cout<<"Hello derive class";
        }
};
```

One More Example of Virtual Keyword-2

```
int main()
 {
        A aobj;
        B bobj;
        A *bptr;
        bptr=&aobj;
        bptr->show();  // call base class function
        bptr=&bobj;
        bptr->show();  // call derive class function
}
```

```
Hello base class
Hello derive class
```



Object Oriented Programming in C++

Your Task.. ☺

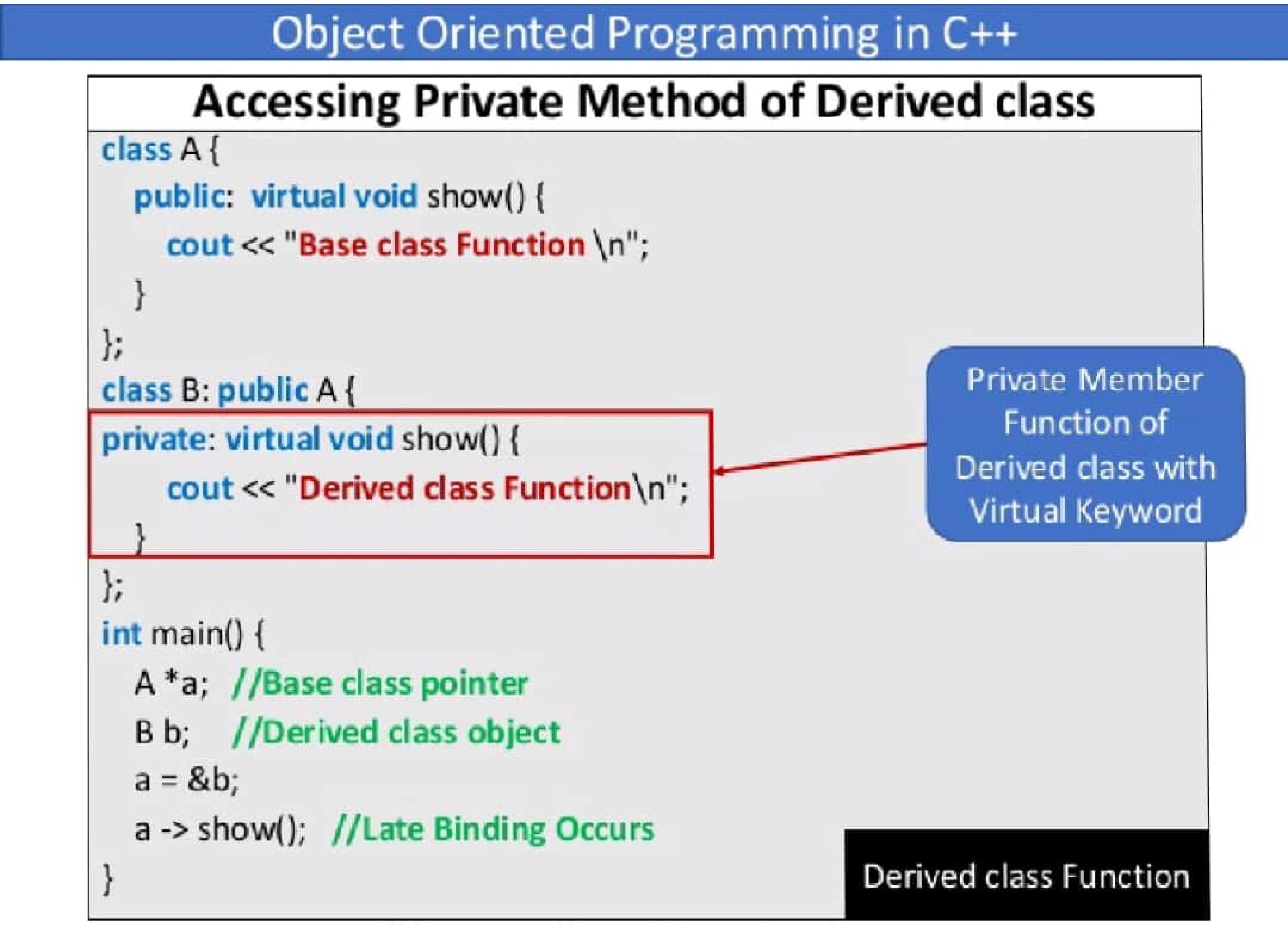
```
class B
{
    public:
     virtual void display()      /* Virtual function */
        { cout<<"Content of base class.\n"; }
};

class D1 : public B
{
    public:
      void display()
        { cout<<"Content of first derived class.\n"; }
};

class D2 : public B
{
    public:
      void display()
        { cout<<"Content of second derived class.\n"; }
};

int main()
{
    B *b;
    D1 d1;
    D2 d2;

/* b->display();  // You cannot use this code here because the function of base class is virtual. */

    b = &d1;
    b->display();   /* calls display() of class derived D1 */
    b = &d2;
    b->display();   /* calls display() of class derived D2 */
    return 0;
}
```

Another use of Virtual Keyword

- **Using Virtual Keyword and Accessing Private Method of Derived class**
- We can call **private** function of derived class from the base class pointer with the help of virtual keyword. Compiler checks for access specifier only at compile time. So at run time when late binding occurs it does not check whether we are calling the private function or public function.

Accessing Private Method of Derived class

```
class A {
   public:  virtual void show() {
      cout << "Base class Function \n";
   }
};
class B: public A {
private: virtual void show() {
      cout << "Derived class Function\n";
   }
};
int main() {
   A *a;  //Base class pointer
   B b;   //Derived class object
   a = &b;
   a -> show();  //Late Binding Occurs
}
```



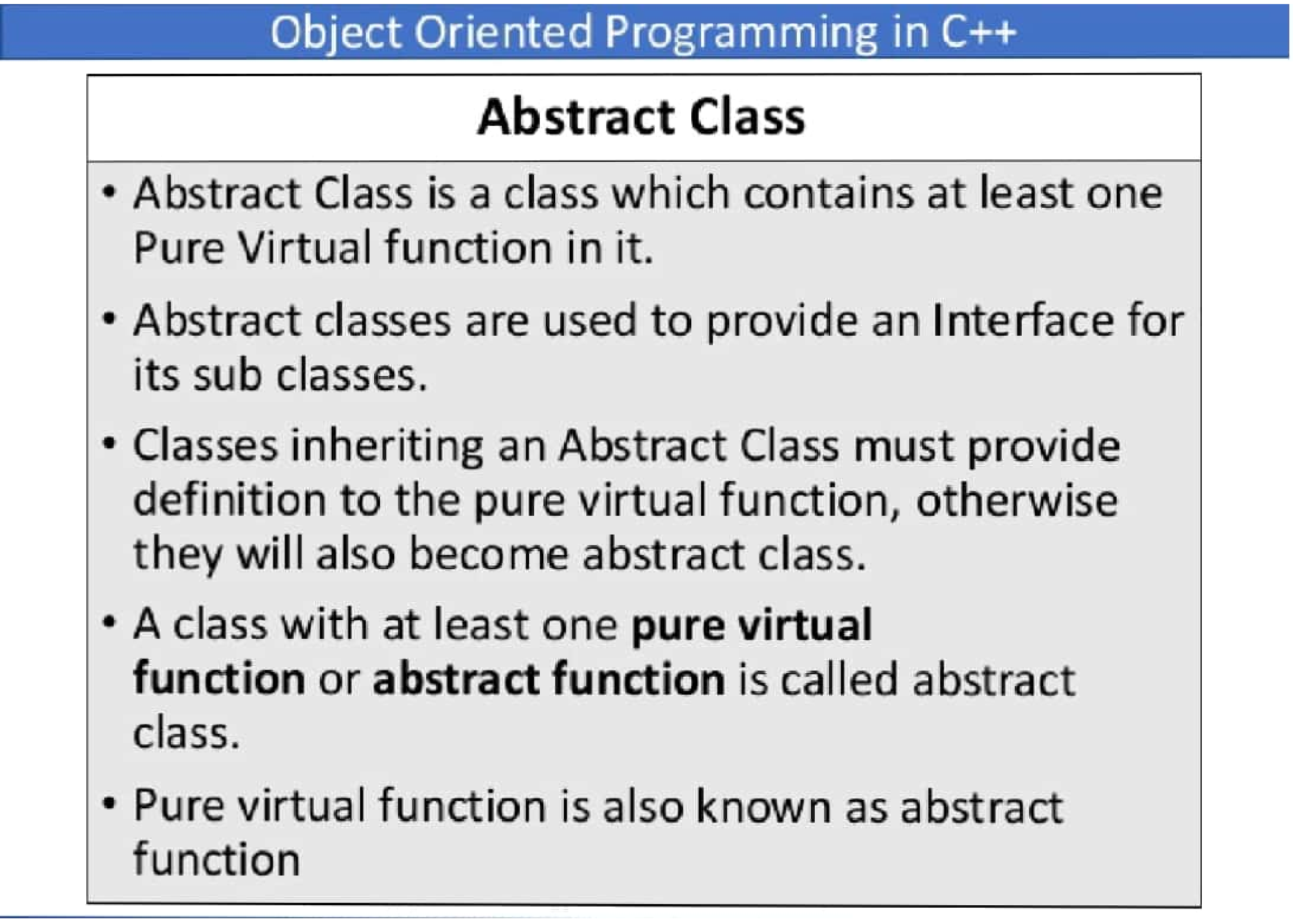
Object Oriented Programming in C++

Accessing Private Method of Derived class

```
class A {
   public:  virtual void show() {
      cout << "Base class Function \n";
   }
};
class B: public A {
private: virtual void show() {
      cout << "Derived class Function\n";
   }
};
int main() {
   A *a;   //Base class pointer
   B b;    //Derived class object
   a = &b;
   a -> show();   //Late Binding Occurs
}
```

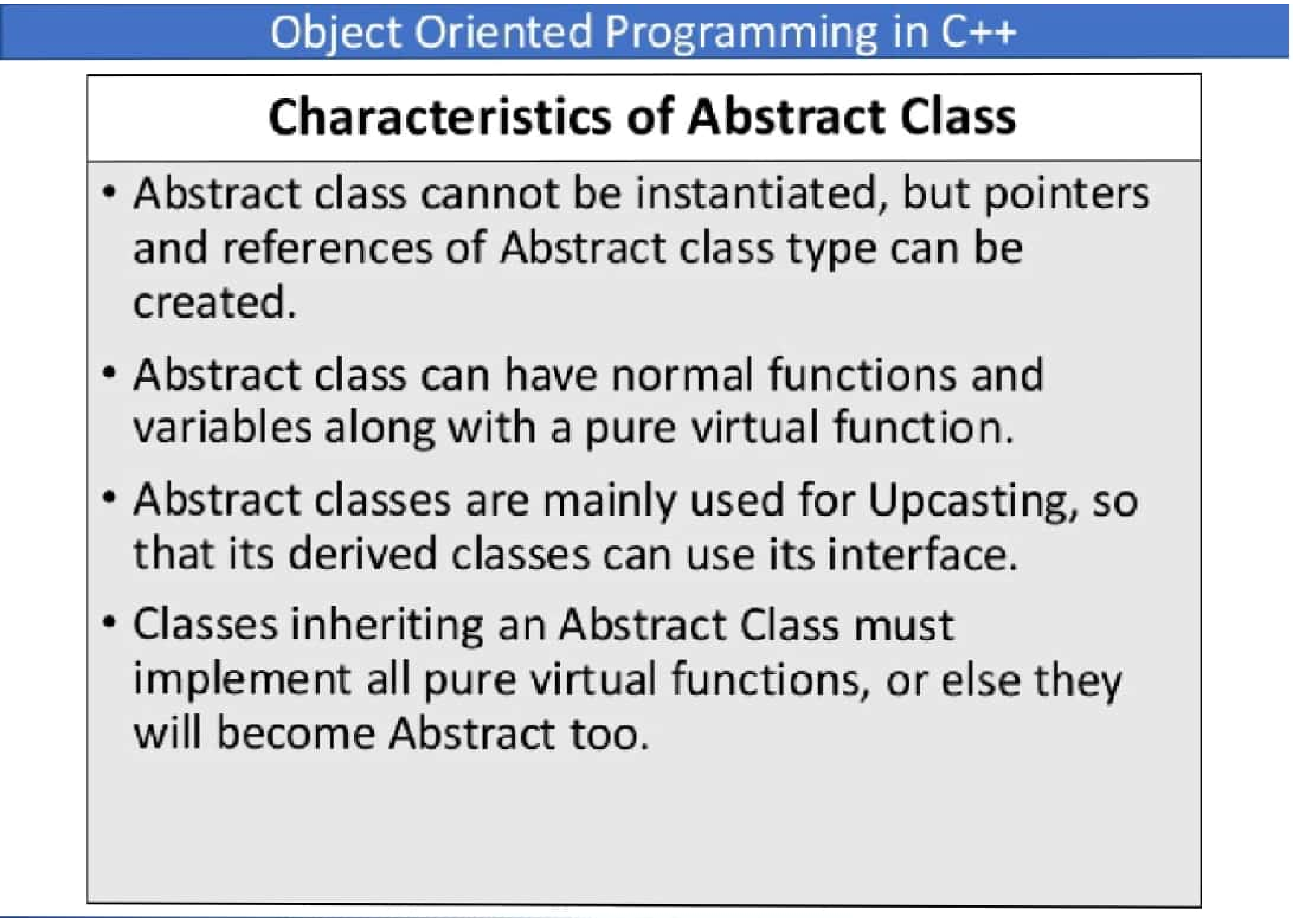
Private Member Function of Derived class with Virtual Keyword

```
Derived class Function
```



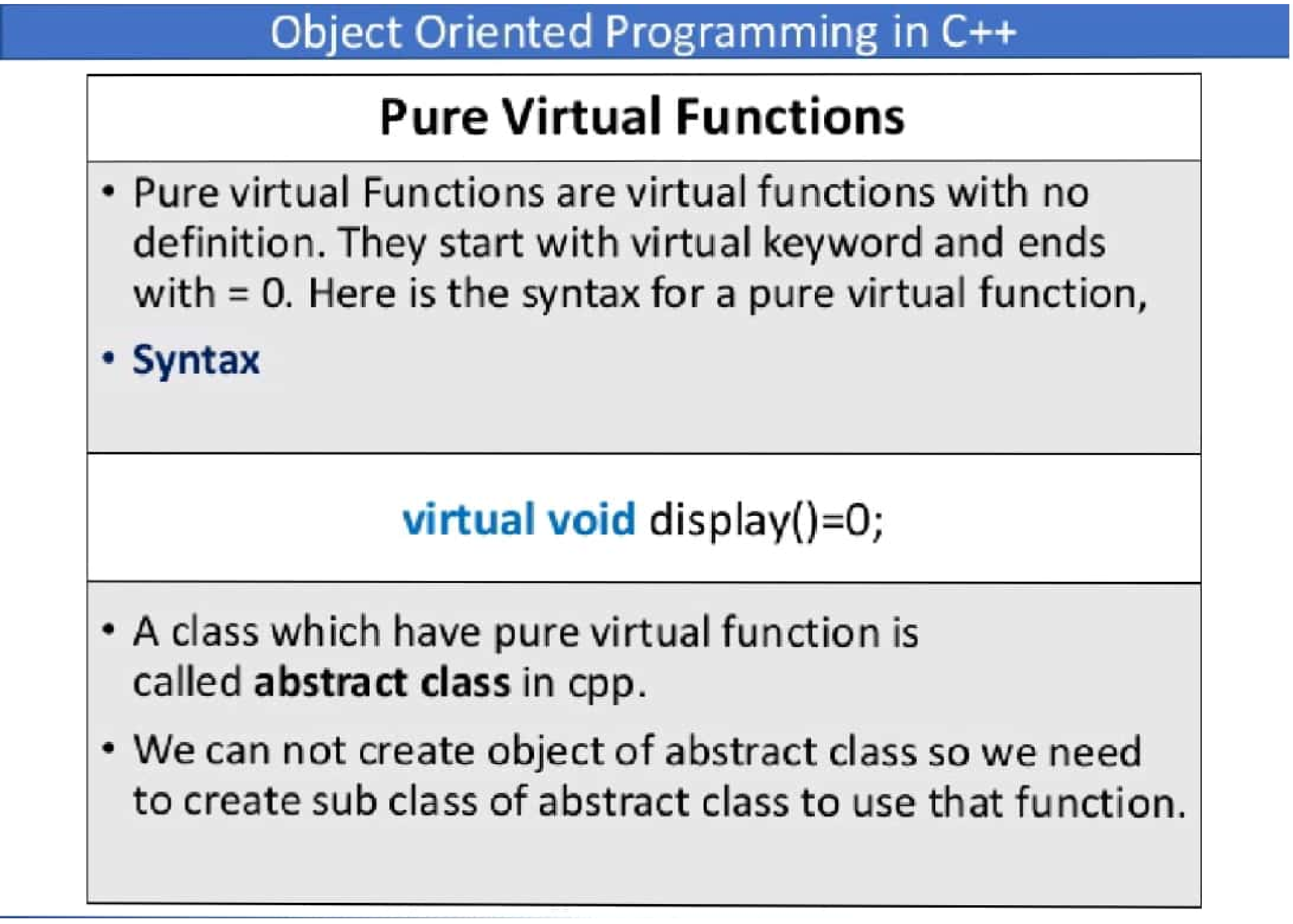
Abstract Class

- Abstract Class is a class which contains at least one Pure Virtual function in it.
- Abstract classes are used to provide an Interface for its sub classes.
- Classes inheriting an Abstract Class must provide definition to the pure virtual function, otherwise they will also become abstract class.
- A class with at least one **pure virtual function** or **abstract function** is called abstract class.
- Pure virtual function is also known as abstract function



Characteristics of Abstract Class

- Abstract class cannot be instantiated, but pointers and references of Abstract class type can be created.
- Abstract class can have normal functions and variables along with a pure virtual function.
- Abstract classes are mainly used for Upcasting, so that its derived classes can use its interface.
- Classes inheriting an Abstract Class must implement all pure virtual functions, or else they will become Abstract too.



Pure Virtual Functions

- Pure virtual Functions are virtual functions with no definition. They start with virtual keyword and ends with = 0. Here is the syntax for a pure virtual function,
- **Syntax**

```
virtual void display()=0;
```

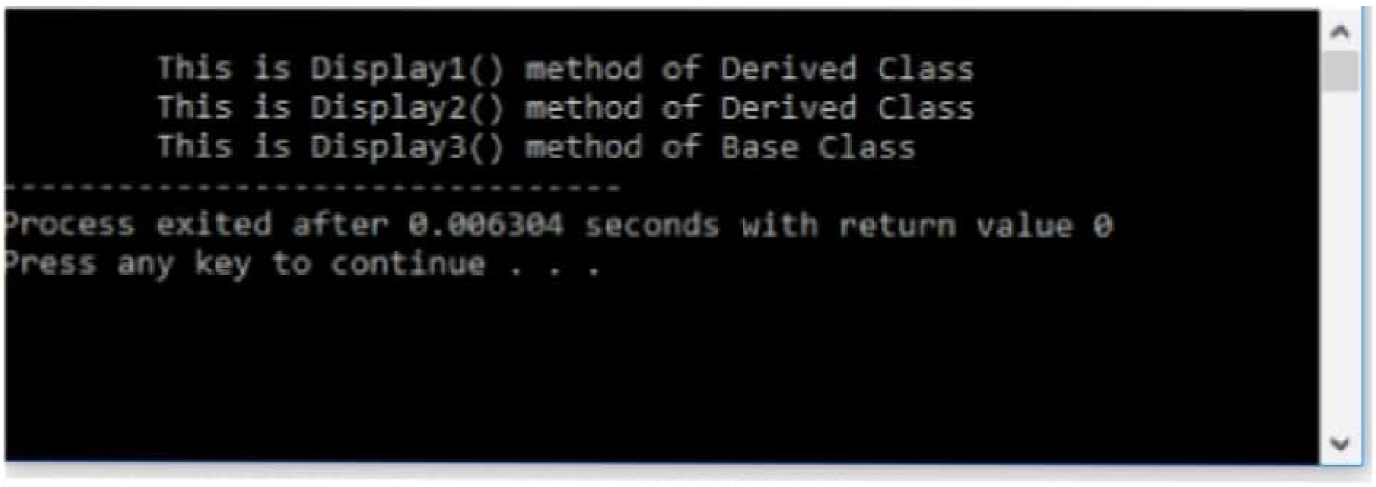
- A class which have pure virtual function is called **abstract class** in cpp.
- We can not create object of abstract class so we need to create sub class of abstract class to use that function.

Example of Pure Virtual Function-1

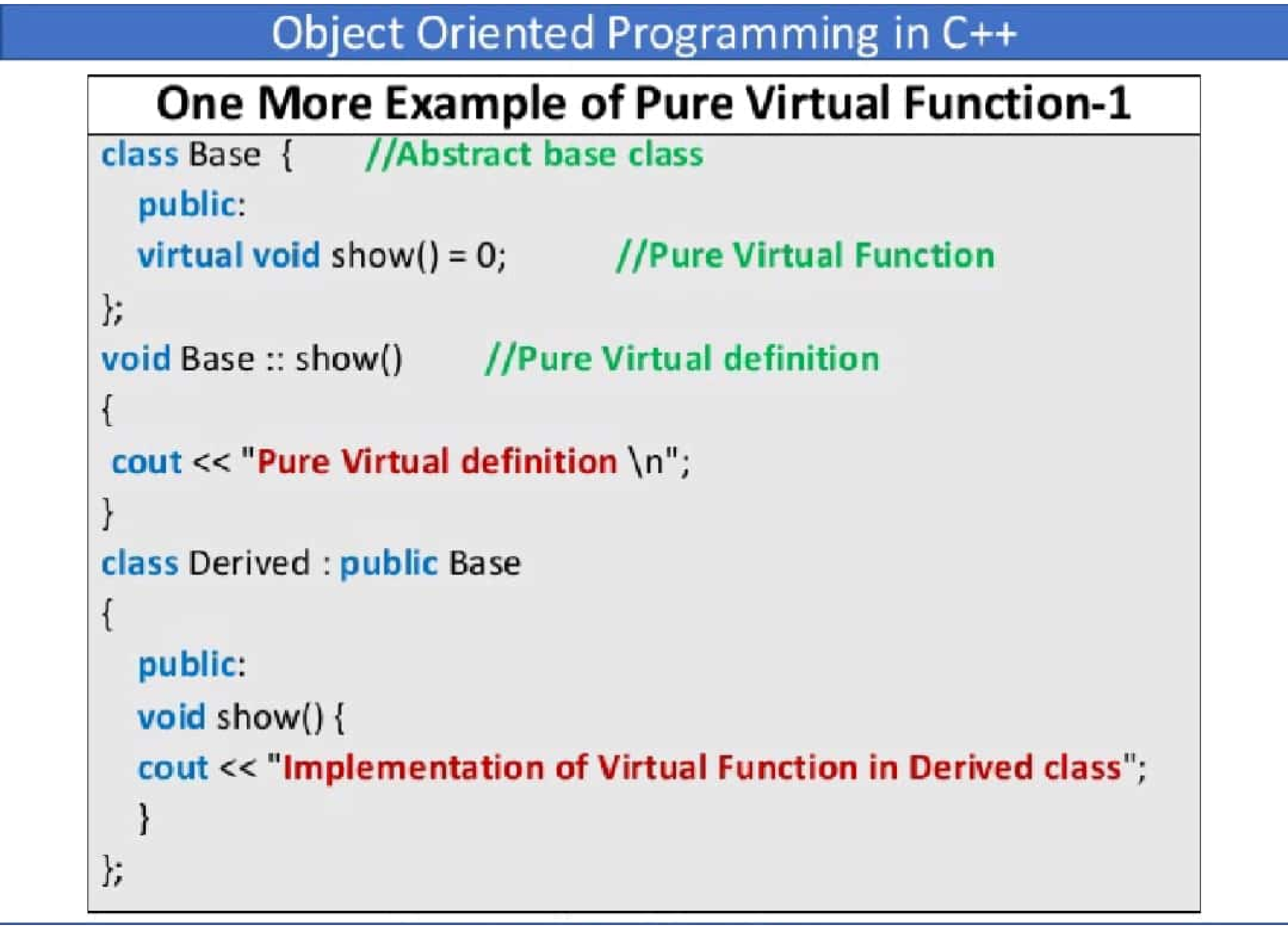
```
class BaseClass {    //Abstract class
   public:
   virtual void Display1()=0;   //Pure virtual function or abstract function
   virtual void Display2()=0;   //Pure virtual function or abstract function
   void Display3() {
      cout<<"\n\tThis is Display3() method of Base Class";
   }
};
class DerivedClass : public BaseClass {
   public:
   void Display1() {
      cout<<"\n\tThis is Display1() method of Derived Class";
   }
    void Display2() {
      cout<<"\n\tThis is Display2() method of Derived Class";
      }
};
```

Example of Pure Virtual Function-2

```
int main()
 {


   DerivedClass D;
   // This will invoke Display1() method of Derived Class
   D.Display1();
   // This will invoke Display2() method of Derived Class
   D.Display2();
  // This will invoke Display3() method of Base Class
   D.Display3();
return 0;
     }
```



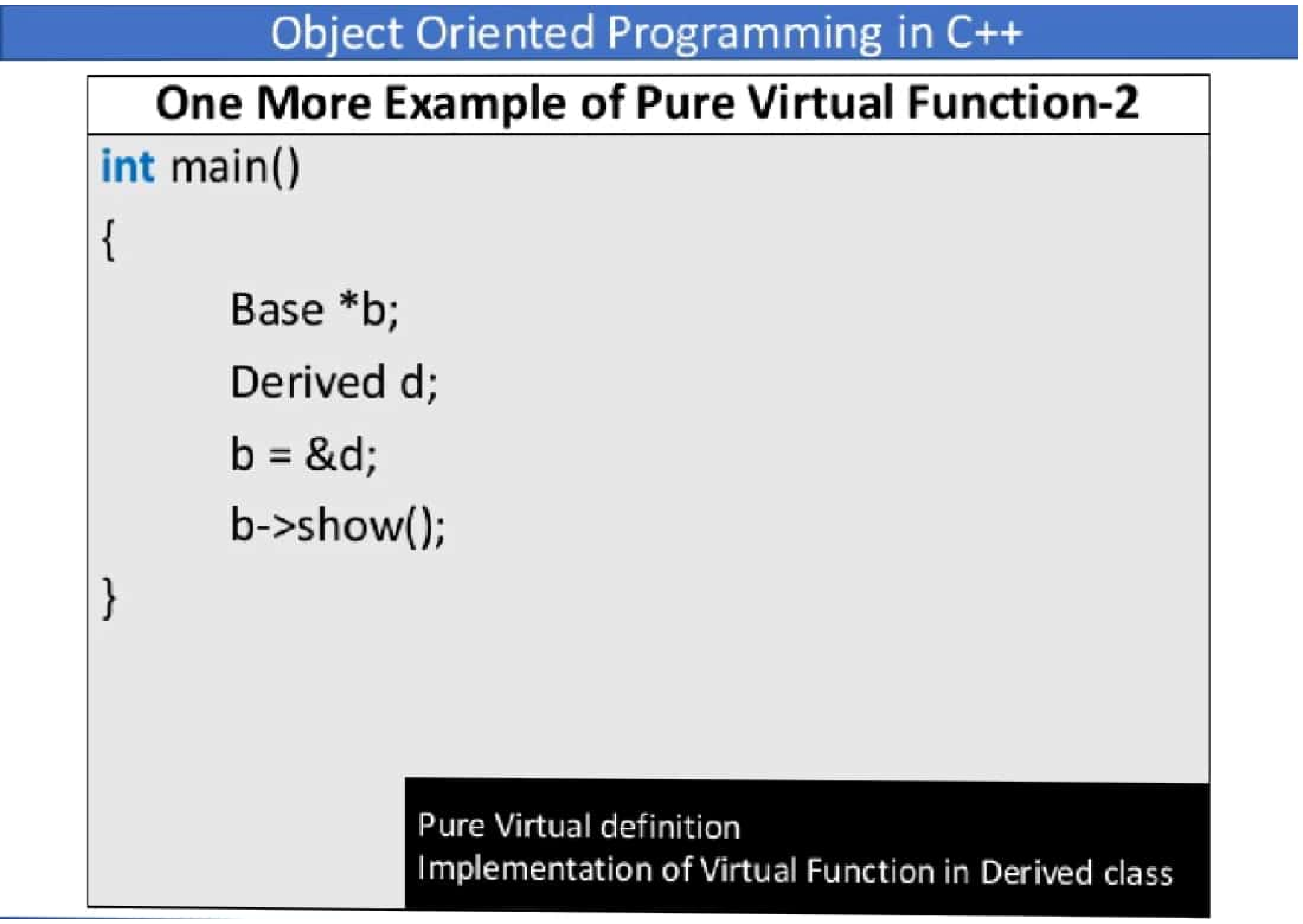
```
        This is Display1() method of Derived Class
        This is Display2() method of Derived Class
        This is Display3() method of Base Class
--------------------------------
Process exited after 0.006304 seconds with return value 0
Press any key to continue . . .
```



Object Oriented Programming in C++

One More Example of Pure Virtual Function-1

```
class Base  {       //Abstract base class
   public:
   virtual void show() = 0;          //Pure Virtual Function
};
void Base :: show()        //Pure Virtual definition
{
 cout << "Pure Virtual definition \n";
}
class Derived : public Base
{
   public:
   void show() {
   cout << "Implementation of Virtual Function in Derived class";
   }
};
```



One More Example of Pure Virtual Function-2

```
int main()
{
        Base *b;
        Derived d;
        b = &d;
        b->show();
}
```

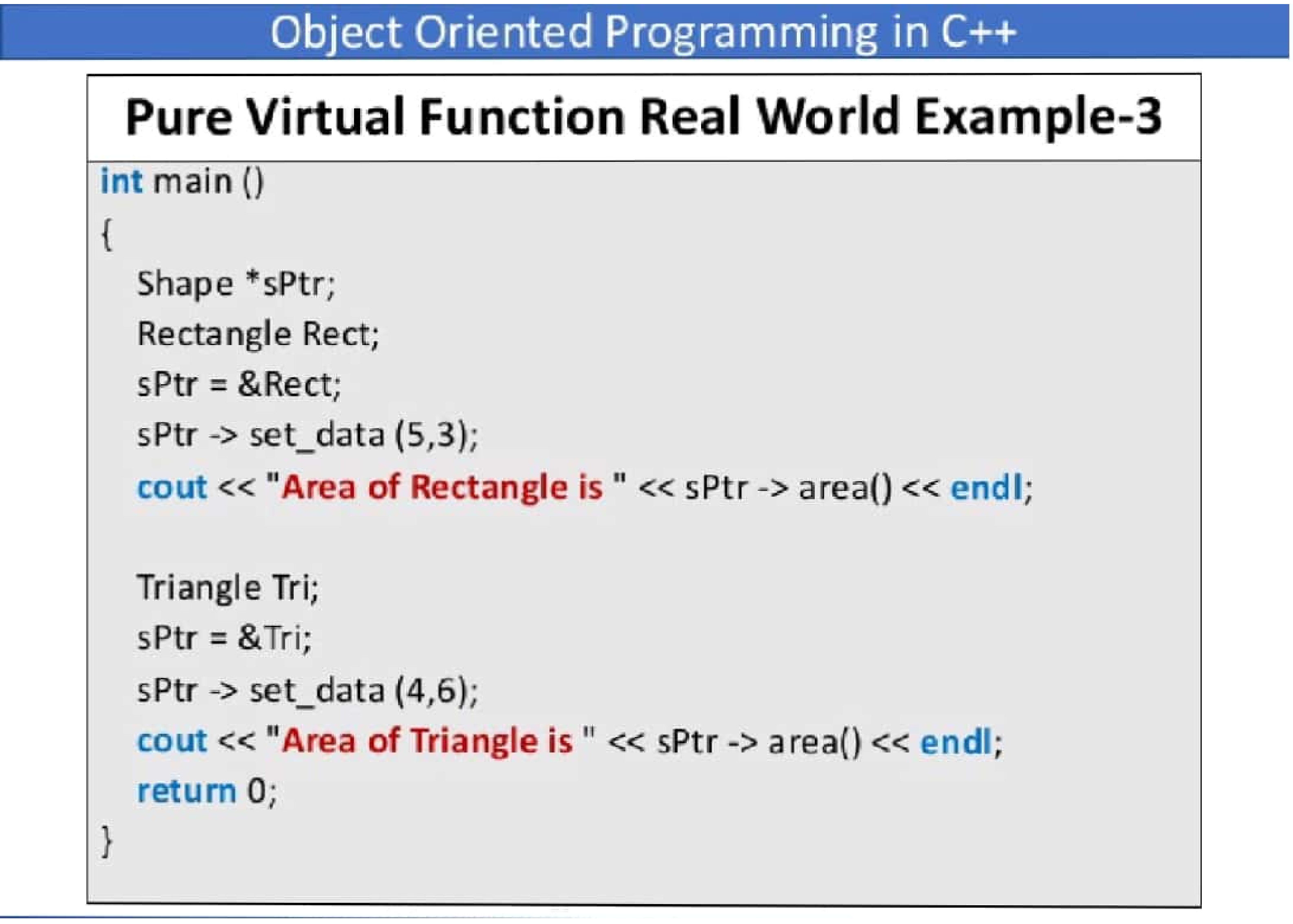
Pure Virtual definition
Implementation of Virtual Function in Derived class

Pure Virtual Function Real World Example-1

```
class Shape
{
  protected:
   double width, height;
  public:
   void set_data (double a, double b)
   {
      width = a;
      height = b;
   }
   virtual double area() = 0;
};
```

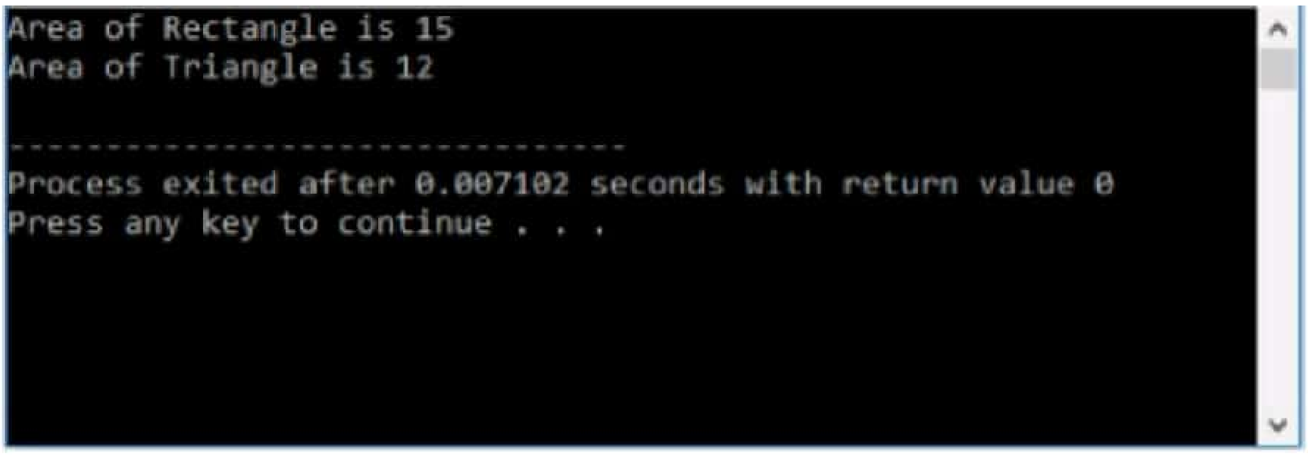
Pure Virtual Function Real World Example-2

```
class Rectangle: public Shape {
public:
   double area ()
   {
      return (width * height);
   }
};
class Triangle: public Shape
{
public:
   double area ()
   {
      return (width * height)/2;
   }
};
```



Pure Virtual Function Real World Example-3

```
int main ()
{
   Shape *sPtr;
   Rectangle Rect;
   sPtr = &Rect;
   sPtr -> set_data (5,3);
   cout << "Area of Rectangle is " << sPtr -> area() << endl;

   Triangle Tri;
   sPtr = &Tri;
   sPtr -> set_data (4,6);
   cout << "Area of Triangle is " << sPtr -> area() << endl;
   return 0;
}
```



```
Area of Rectangle is 15
Area of Triangle is 12

--------------------------------
Process exited after 0.007102 seconds with return value 0
Press any key to continue . . .
```