

<pre> #ifndef BASE_H #define BASE_H class Base { public: Base (int = 0); void Display () const; virtual void Print () const; private: int x; }; #endif </pre>	<pre> #include <iostream> using namespace std; #include "Base.h" Base::Base (int a) : x (a) { } void Base::Display () const { cout<<"Dispaly (): Base class\n x = "<<x<<endl; } void Base::Print () const { cout<<"Print (): Base class\n x = "<<x<<endl; } </pre>
---	--

<pre> #ifndef DERIVED_H #define DERIVED_H #include "Base.h" class Derived : public Base { public: Derived (int = 0, int = 0); void Display () const; virtual void Print () const; private: int y; }; #endif </pre>	<pre> #include <iostream> using namespace std; #include "Derived.h" Derived::Derived (int a, int b) : Base (a), y (b) { } void Derived::Display () const { Base::Display (); cout<<"Dispaly (): Derived class\n y = "<<y<<endl; } void Derived::Print () const { Base::Print (); cout<<"Print (): Derived class\n y = "<<y<<endl; } </pre>
---	--

```

#include "Base.h"
#include "Derived.h"

void CallValue (Base b);
void CallReference (Base& b);
void CallPointer (Base* b);

void main ()
{
    Base b (5);
    Derived d (10, 20);

    CallValue (b);
    CallValue (d);

    CallReference (b);
    CallReference (d);

    Base* bptr = new Base (5);
    Derived* dptr = new Derived (10, 20);

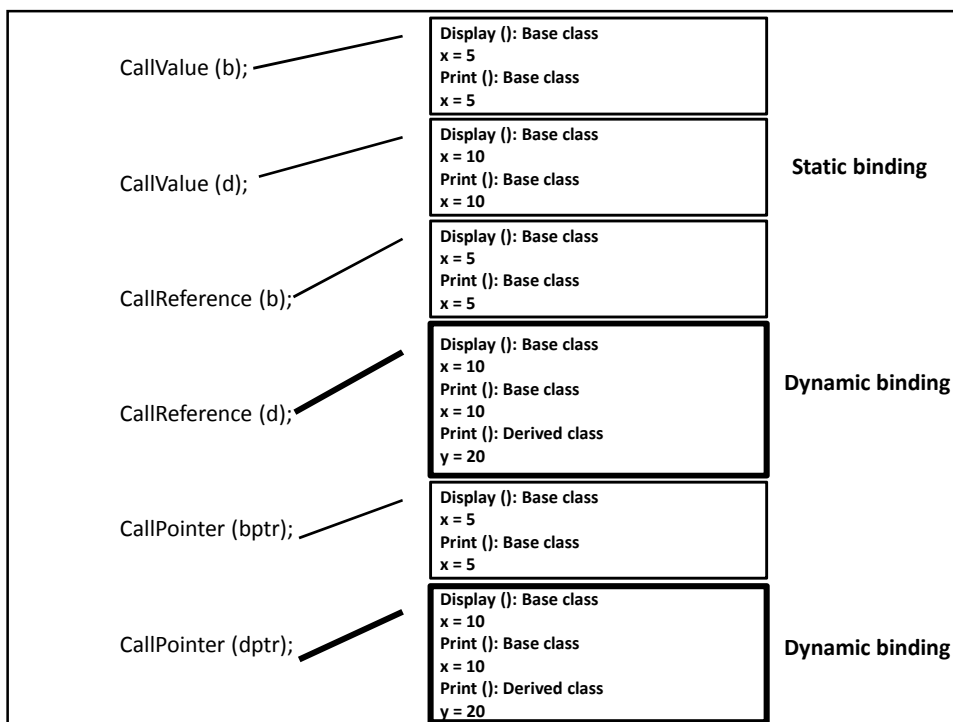
    CallPointer (bptr);
    CallPointer (dptr);
}

void CallValue (Base b)
{
    b.Display ();
    b.Print ();
}

void CallReference (Base& b)
{
    b.Display ();
    b.Print ();
}

void CallPointer (Base* b)
{
    b->Display ();
    b->Print ();
}

```



Static vs Dynamic Binding

- In **compile-time binding**, the necessary code to call a specific function is generated by the compiler. (Compile-time binding is also known as **static binding** or **early binding**.)
- The binding of virtual functions occurs at program execution time, not at compile time. This kind of binding is called **run-time binding** or **late binding**.

Abstract Class

- Abstract class has at least one pure virtual function
- A pure virtual function has initializer = **0**
e.g. `virtual void Print () const = 0;`
- A pure virtual function doesn't have implementation and must be overridden in derived classes
- No objects can be created of an Abstract class
- Abstract classes are merely used as base classes for some other classes.

```
class Number
{
public:
    virtual void print () const = 0;
};
```

```
class IntNumber : public Number
{
public:
    IntNumber (int a)
        : x (a)
    {}
    virtual void print () const
    { cout<<x<<endl;}

private:
    int x;
};
```

```
class FloatNumber : public Number
{
public:
    FloatNumber (float a)
        : x (a)
    {}
    virtual void print () const
    { cout<<x<<endl;}

private:
    float x;
};
```

```
void main ()
{
    Number* numbers[3];

    IntNumber* i1 = new IntNumber (5);
    FloatNumber* f1 = new FloatNumber (10.7f);
    IntNumber* i2 = new IntNumber (8);

    numbers[0] = i1;
    numbers[1] = f1;
    numbers[2] = i2;

    for (int i = 0; i < 3; ++i)
        numbers[i]->print ();
}
```