



Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

Module 27: Programming in C++

Polymorphism: Part 2: Static and Dynamic Binding

Instructors: Abir Das and Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

{abir, sourangshu}@cse.iitkgp.ac.in

Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**



Module Objectives

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

- Understand Static and Dynamic Binding
- Understand Polymorphic Type



Module Outline

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

- 1 Type Binding
 - Type of an Object
 - Static and Dynamic Binding
 - Comparison of Static and Dynamic Binding
 - Static Binding
 - Dynamic Binding
- 2 Polymorphic Type
- 3 Module Summary



Type of an Object

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

- The *static type* of the object is the type declared for the object while writing the code
- Compiler *sees static type*
- The *dynamic type* of the object is determined by the type of the object to which it *refers at run-time*
- Compiler *does not see dynamic type*

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

```
int main() {  
    A *p;  
    p = new B; // Static type of p is A*  
               // Dynamic type of p is B*  
}
```



Static and Dynamic Binding

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

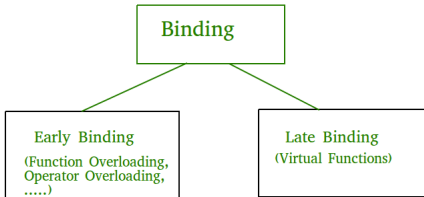
Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

- **Static binding (early binding)**: When a function invocation binds to the function definition based on the static type of objects
 - This is done at *compile-time*
 - Normal *function calls*, *overloaded function calls*, and *overloaded operators* are examples of *static binding*
- **Dynamic binding (late binding)**: When a function invocation binds to the function definition based on the dynamic type of objects
 - This is done at *run-time*
 - *Function pointers*, *Virtual functions* are examples of *late binding*





Comparison of Static and Dynamic Binding

Basis	Static Binding	Dynamic Binding
<ul style="list-style-type: none">● Event Occurrence● Information● Advantage● Time● Actual Object● Alternate name● Example	<ul style="list-style-type: none">● Events occur at <i>compile time</i> – <i>Static Binding</i>● All information needed to call a function is known at <i>compile time</i><ul style="list-style-type: none">● <i>Efficiency</i>● <i>Fast execution</i>● Actual object is <i>not used for binding</i>● <i>Early Binding</i>● <i>Method Overloading</i> Normal function call, Overloaded function call, Overloaded operators	<ul style="list-style-type: none">● Events occur at <i>run time</i> – <i>Dynamic Binding</i>● All information needed to call a function is known only at <i>run time</i><ul style="list-style-type: none">● <i>Flexibility</i>● <i>Slow execution</i>● Actual object is <i>used for binding</i>● <i>Late Binding</i>● <i>Method Overriding</i> Virtual functions



Static Binding

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

Inherited Method

```
#include<iostream>
using namespace std;
class B { public:
    void f() { }
};
class D : public B { public:
    void g() { } // new function
};
int main() { B b; D d;

    b.f(); // B::f()
    d.f(); // B::f() ----- Inherited
    d.g(); // D::g() ----- Added
}
```

- Object **d** of derived class inherits the base class function **f()** and has its own function **g()**
- Function calls are resolved at compile time based on static type

Overridden Method

```
#include<iostream>
using namespace std;
class B { public:
    void f() { }
};
class D : public B { public:
    void f() { }
};
int main() { B b; D d;

    b.f(); // B::f()
    d.f(); // D::f() ----- Overridden
                        // masks the base class function
}
```

- If a member function of a base class is redefined in a derived class with the same signature then it masks the base class method
- The derived class method **f()** is linked to the object **d**. As **f()** is redefined in the derived class, the base class version cannot be called with the object of a derived class



Member Functions: Overrides and Overloads: RECAP (Module 22)

Module 22

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

Inheritance

```
class B { public: // Base Class
    void f(int i);
    void g(int i);
};
class D: public B { public: // Derived Class
    // Inherits B::f(int)

    // Inherits B::g(int)
};
B b;
D d;

b.f(1); // Calls B::f(int)
b.g(2); // Calls B::g(int)

d.f(3); // Calls B::f(int)
d.g(4); // Calls B::g(int)
```

- `D::f(int)` overrides `B::f(int)`
- `D::f(string&)` overloads `B::f(int)`

Override & Overload

```
class B { public: // Base Class
    void f(int);
    void g(int i);
};
class D: public B { public: // Derived Class
    // Inherits B::f(int)
    void f(int); // Overrides B::f(int)
    void f(string&); // Overloads B::f(int)
    // Inherits B::g(int)
    void h(int i); // Adds D::h(int)
};
B b;
D d;

b.f(1); // Calls B::f(int)
b.g(2); // Calls B::g(int)

d.f(3); // Calls D::f(int)
d.g(4); // Calls B::g(int)

d.f("red"); // Calls D::f(string&)
d.h(5); // Calls D::h(int)
```




using Construct – Avoid Method Hiding

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

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

class A { public:
    void f() { }
};

class B : public A { public:
    // To overload, rather than hide the base class function f(),
    // it is introduced into the scope of B with a using declaration
    using A::f;
    void f(int) { } // Overloads f()
};

int main() {
    B b; // function calls resolved at compile time

    b.f(3); // B::f(int)
    b.f();  // A::f()
}
```

- Object `b` of derived class linked to with inherited base class function `f()` and the overloaded version defined by the derived class `f(int)`, based on the input parameters – function calls resolved at compile time



Dynamic Binding

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

Non-Virtual Method

```
#include<iostream>
using namespace std;
class B { public:
    void f() { }
};
class D : public B { public:
    void f() { }
};
int main() {
    B b;
    D d;

    B *p;

    p = &b; p->f(); // B::f()
    p = &d; p->f(); // B::f()
}
```

- `p->f()` always binds to `B::f()`
- Binding is decided by the *type of pointer*
- **Static Binding**

Virtual Method

```
#include<iostream>
using namespace std;
class B { public:
    virtual void f() { }
};
class D : public B { public:
    virtual void f() { }
};
int main() {
    B b;
    D d;

    B *p;

    p = &b; p->f(); // B::f()
    p = &d; p->f(); // D::f()
}
```

- `p->f()` binds to `B::f()` for a B object, and to `D::f()` for a D object
- Binding is decided by the *type of object*
- **Dynamic Binding**



Static and Dynamic Binding

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

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

class B { public:
    void f() { cout << "B::f()" << endl; }
    virtual void g() { cout << "B::g()" << endl; }
};

class D: public B { public:
    void f() { cout << "D::f()" << endl; }
    virtual void g() { cout << "D::g()" << endl; }
};

int main() { B b; D d;

    B *pb = &b;
    B *pd = &d; // UPGRADE

    B &rb = b;
    B &rd = d; // UPGRADE

    b.f(); // B::f()
    b.g(); // B::g()
    d.f(); // D::f()
    d.g(); // D::g()

    pb->f(); // B::f() -- Static Binding
    pb->g(); // B::g() -- Dynamic Binding
    pd->f(); // B::f() -- Static Binding
    pd->g(); // D::g() -- Dynamic Binding

    rb.f(); // B::f() -- Static Binding
    rb.g(); // B::g() -- Dynamic Binding
    rd.f(); // D::f() -- Static Binding
    rd.g(); // D::g() -- Dynamic Binding

    return 0;
}
```



Polymorphic Type: Virtual Functions

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

- *Dynamic binding* is possible only for pointer and reference data types and for member functions that are declared as **virtual** in the base class
- These are called **Virtual Functions**
- If a member function is declared as virtual, it can be overridden in the derived class
- If a member function is not virtual and it is re-defined in the derived class then the latter definition hides the former one
- Any class containing a virtual member function – by definition or by inheritance – is called a **Polymorphic Type**
- A hierarchy may be *polymorphic* or *non-polymorphic*
- A non-polymorphic hierarchy has little value



Polymorphism Rule

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

```
#include <iostream>
using namespace std;
class A { public:
    void f()      { cout << "A::f()" << endl; } // Non-Virtual
    virtual void g() { cout << "A::g()" << endl; } // Virtual
    void h()      { cout << "A::h()" << endl; } // Non-Virtual
};

class B : public A { public:
    void f()      { cout << "B::f()" << endl; } // Non-Virtual
    void g()      { cout << "B::g()" << endl; } // Virtual
    virtual void h() { cout << "B::h()" << endl; } // Virtual
};

class C : public B { public:
    void f()      { cout << "C::f()" << endl; } // Non-Virtual
    void g()      { cout << "C::g()" << endl; } // Virtual
    void h()      { cout << "C::h()" << endl; } // Virtual
};

int main() {
    B *q = new C; A *p = q;

    p->f();
    p->g();
    p->h();
}
```

	q->f();	A::f()
	q->g();	C::g()
	q->h();	A::h()
		B::f()
		C::g()
		C::h()



Module Summary

Module 27

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Type Binding

Type of an Object

Static and Dynamic
Binding

Comparison

Static Binding

Dynamic Binding

Polymorphic Type

Module Summary

- Discussed Static and Dynamic Binding
- Polymorphic type introduced