19CSE201 :Advanced Programming

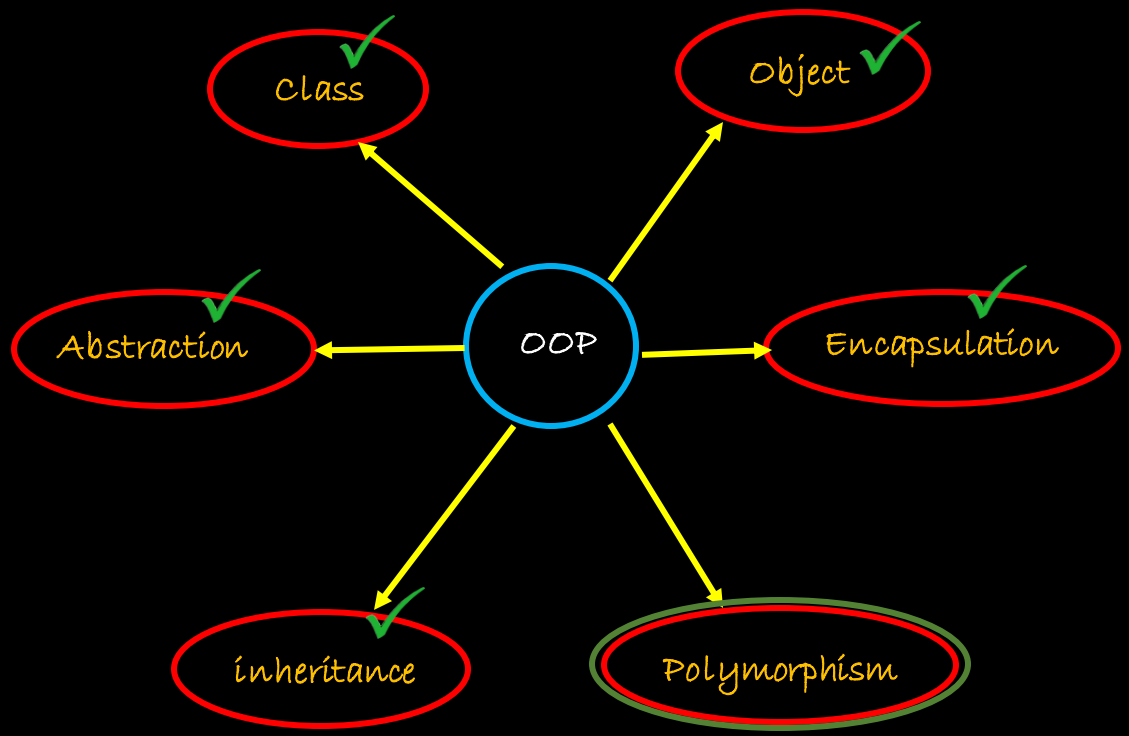
Lecture 15
More on Polymorphism in C++

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg
Amrita Vishwa Vidyapeetham - Coimbatore

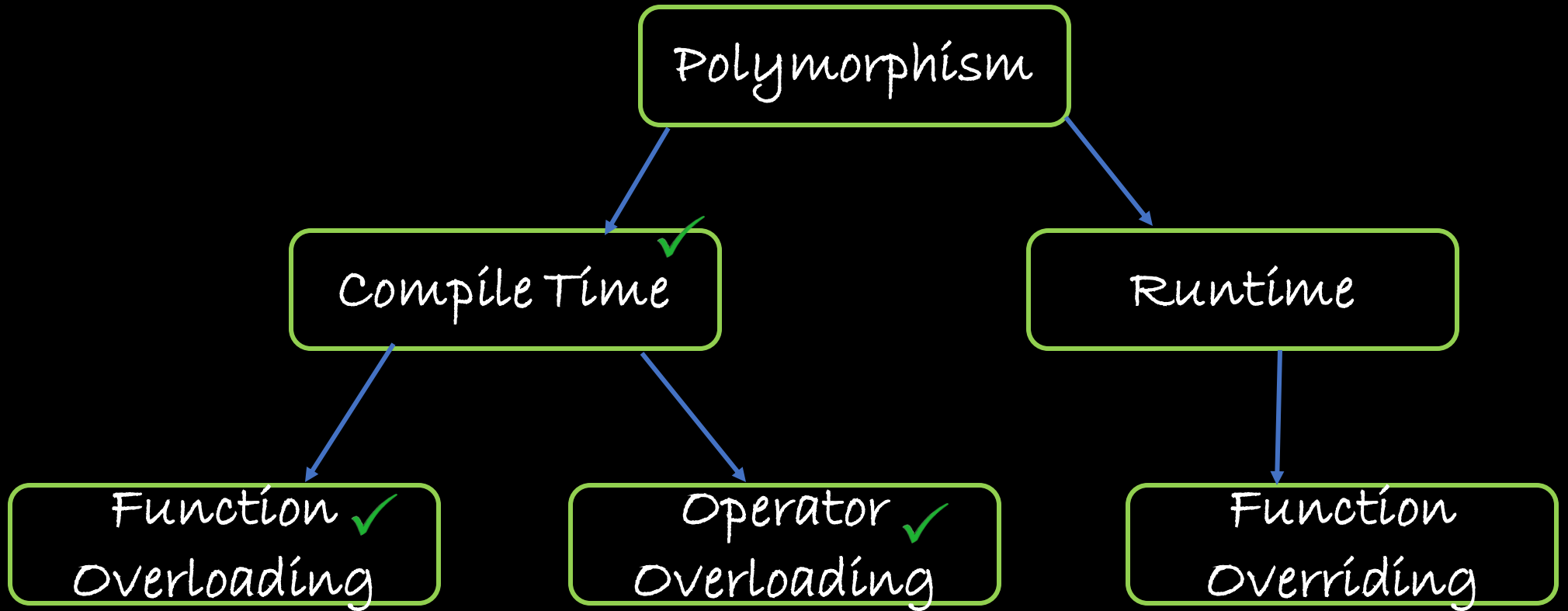
Based on material from Stroustrup B. Programming: principles and practice using C++. Second edition, Addison Wesley; 2014 and multiple sources on the internet

A Quick Recap

- OOP Overview
- Polymorphism
- Function/method overloading
- Operator Overloading
- Examples & Exercises



Types of Polymorphism



Runtime Polymorphism

- Runtime polymorphism is also known as dynamic polymorphism or late binding or dynamic binding.
- In runtime polymorphism, the function call is resolved at run time.
- Example: Function Overriding
- Note on Binding:
  - For every function call, the compiler binds or links the call to the corresponding function definition.
  - This linking can happen at 2 different times
    - At the time of compiling program – Compile Time / Early Binding
    - During program execution - Runtime / Late Binding

Runtime Polymorphism - Overriding

- Function overriding is a feature that allows us to have a same function in child class which is already present in the parent class.
- It is like creating a new version of an old function, in the child class.
- To override a function you must have the same signature in child class.
- The function in parent class is called the overridden function and function in child class is called overriding function.

Function Overriding Cont.

- In case of function overriding we have two definitions of the same function
  - one is in the parent class and one in child class.
- The call to the function is determined at runtime to decide which definition of the function is to be called
  - Hence it is "Runtime Polymorphism"

Overriding - Example

```
class A {
public:
  void disp(){
     cout<<"Super Class Function";
  }
};
```

```
class B: public A{
public:
  void disp(){
     cout<<"Sub Class Function";
     A :: disp();
  }
};
```

```
int main() {
    A obj; //Parent class object
  obj.disp();  //Child class object
  B obj2;
  obj2.disp();
  return 0;
}
```

Overriding – Example Cont. Try these!

```
//Example2
int main() {
  A obj;
  obj.disp();
  B obj2;
  obj2.disp();
  A obj3 = B();
  Obj3.disp();
  return 0;
}
```

```
//Example3
int main() {
  A obj;
  obj.disp();
  B obj2;
  obj2.disp();
  Obj2.A::disp();
  return 0;
}
```

Overriding – Another Example

```
class A{
public:
  void disp(){
     cout<<"Super Class Function";
  }
};
```

```
class B: public A{
public:
  void disp(){
     cout<<"Sub Class Function";
      }
};
```

```
//main 1
int main() {
  B obj2;
  obj2.disp();
  return 0;
}
```

```
//main 2- using pointers
int main() {
  B obj2;
  // pointer of Base type that points to
  derived1
    A* ptr = &obj2;
// call function of Base class using ptr
    ptr->disp();
  return 0;
}
```

Pointer Behaviour in polymorphism

- A base class pointer variable can hold address of derived class object, but it can access only members of base class.

```
class base
{
    public:
        void show()
        {
            cout<<"from based class"<<endl;
        }
};
class derived:public base
{
        public:
        void show()
        {
            cout<<"from derived class"<<endl;
        }
};
```

```
int main()
{
    base*ptr;
    derived d;
    ptr=&d;
    ptr->show();
    return 0;

}
```

Pointer Behaviour in polymorphism Cont.

- In the previous example you can see that even the pointer holds address of derived class object; it has called the base version of show() method.
- The problem is; even if a base class pointer holds address of derived type of object, it can access only members of base class
  - This is because the base pointer variable doesn't have any idea about the structure of derived class.

Quick Summary

- Polymorphism
- Runtime Polymorphism
- Binding in C++
- Pointer Behaviour in Polymorhism
- Examples
- Exercises

Up Next

Some Special Functions and Classes in C++