

TAKE OUT A PAPER AND SOLVE THE TASK.
IT'S A MARKED ACTIVITY

Suppose you are a programmer working for a software house and you have a base class called Employee with a data member emp_id and member function work(). The work() function performs the tasks assigned to the employee based on their job responsibilities. //just print I am working

You have two types of employees: Developer and Tester. Both types of employees inherit from the Employee base class.

Developer has a member function called write_code() that takes in a programming language parameter and writes code in the specified language (print I am developing). Tester has a member function called test_code() that tests the code(print I am testing).

Make Parameterized constructors for all classes.

POLYMORPHISM



POLYMORPHISM

The process of representing one Form in multiple forms is known as Polymorphism

Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and morphs means forms. So polymorphism means many forms.



REAL LIFE EXAMPLE OF POLYMORPHISM

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you are at your home at that time you behave like a son or daughter, Here one person have different-different behaviors.



In Shopping malls behave like Customer

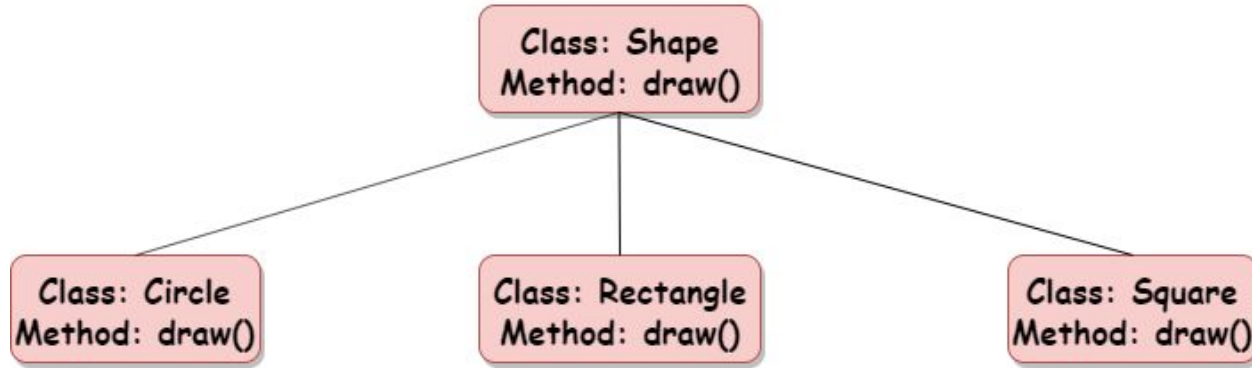
In Bus behave like Passenger

In School behave like Student

At Home behave like Son

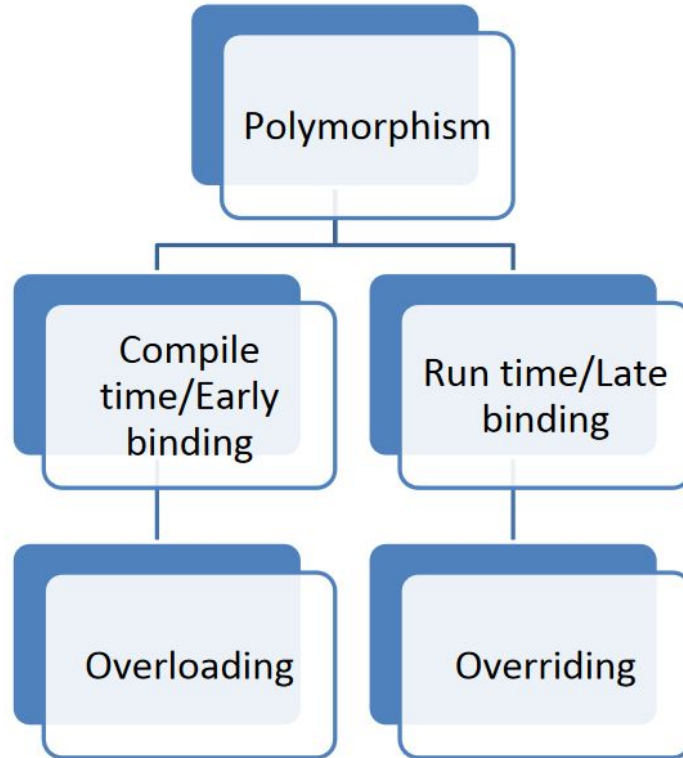
POLYMORPHISM IN OOP

Polymorphism refers to the ability of a method to be used in different ways, that is, it can take different forms at different times (poly + morphos).

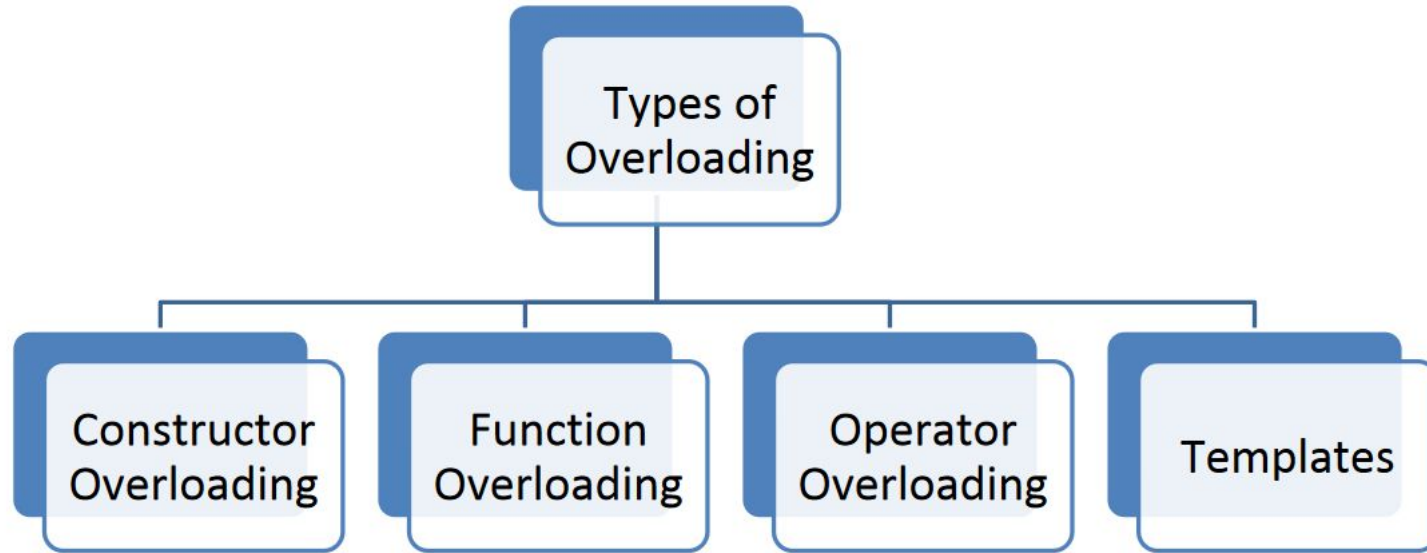


Polymorphism enables us to “program in the general” rather than “program in the specific.”

TYPES OF POLYMORPHISM



COMPILE TIME / STATIC / EARLY BINDING POLYMORPHISM



STATIC / COMPILE TIME POLYMORPHISM

It happens where more than one methods share the same name with different parameters or signature and different return type.

It is known as Early Binding because the compiler is aware of the functions with same name and also which overloaded function is to be called is known at compile time.

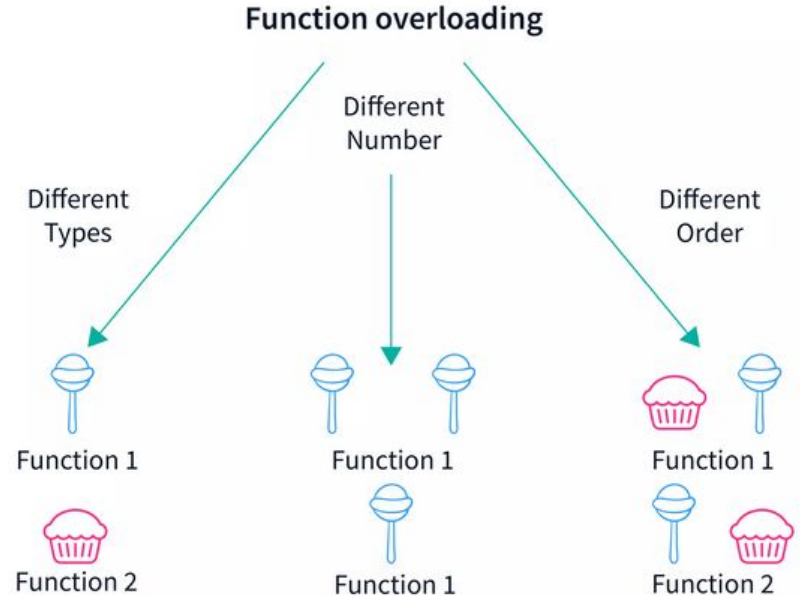
CONSTRUCTOR OVERLOADING

We discussed this already.

FUNCTION OVERLOADING

C++ enables several functions of the same name to be defined, as long as they have different parameter signatures.

The compiler selects the proper function to call by examining the number, types and order of the arguments in the call.



FUNCTION OVERLOADING

Function overloading allows the program to follow the principle of “Polymorphism”

The return type or name of parameters don't help the compiler in selecting which overloaded function to call.

```
void findPerson(string name) { ... }
```

```
void findPerson(int ID) { ... }
```

```
void findPerson(int ID, string addr) { ... }
```

```
void findPerson(string addr, int ID) { ... }
```

Overload the function write code with 1 programming language or 2 programming language to develop.

Overload the function test code with 1 programming language or 2 programming language to test.

DEFAULT ARGUMENT

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn't provide a value for the argument with a default value.

Default arguments must be the rightmost (trailing) arguments in a function's parameter list. default arguments are assigned from right to left.

Once we provide a default value for a parameter, all subsequent parameters must also have default values.

// Invalid

```
void add(int a, int b = 3, int c, int d);
```

// Invalid

```
void add(int a, int b = 3, int c, int d = 4);
```

// Valid

```
void add(int a, int c, int b = 3, int d = 4);
```

FUNCTION OVERLOADING

Avoid using default arguments with overloaded functions as it can be dangerous i.e. can lead to error conditions that are hard to trace.

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

class printData {
public:
    void print() {
        cout << "Printing default " << endl;
    }
    void print(float f) {
        cout << "Printing float: " << f << endl;
    }
};

int main(void) {
    printData pd;
    pd.print(); // Call print to print default
    pd.print(5.6); // Call print to print float
    return 0;
}
```

FUNCTION OVERLOADING

Avoid using default arguments with overloaded functions as it can be dangerous i.e. can lead to error conditions that are hard to trace.

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

```
class printData {
public:
    void print() {
        cout << "Printing default " << endl; }
    void print(float f=0) {
        cout << "Printing float: " << f << endl; } };
```

```
int main(void) {
    printData pd;
    pd.print(); //[Error] call of overloaded 'print()' is ambiguous
    pd.print(5.6); // Call print to print float
    return 0; }
```


FUNCTION OVERLOADING

A const member function can be overloaded with a non-const version.

The compiler chooses which overloaded member function to use based on the object on which the function is invoked. If the object is const, the compiler uses the const version. If the object is not const, the compiler uses the non-const version.

FUNCTION OVERLOADING

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

class printData {
public:
void print() {
cout << "Printing default Function" << endl;}
void print() const{
cout << "Printing const Function " << endl;}
};

int main(void) {
printData pd;
const printData p;
pd.print(); // Call print to print default
p.print(); // Call print to const Function
return 0;}
```

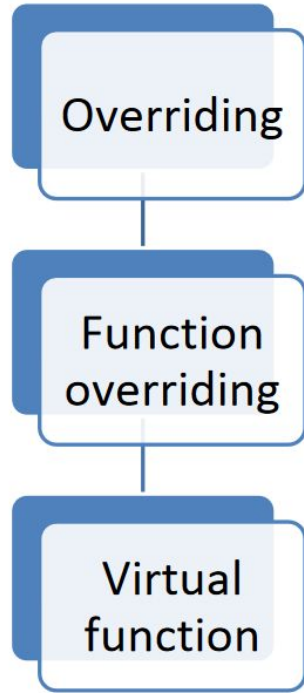
OPERATOR OVERLOADING

We will discuss this next week.

TEMPLATES

We will discuss this later.

RUN TIME / DYNAMIC / LATE BINDING POLYMORPHISM



RUN TIME / DYNAMIC / LATE BINDING POLYMORPHISM

This refers to the entity which changes its form depending on circumstances at runtime. This concept can be adopted as analogous to a chameleon changing its color at the sight of an approaching object.

Method Overriding uses runtime Polymorphism.

It is also called Late Binding.

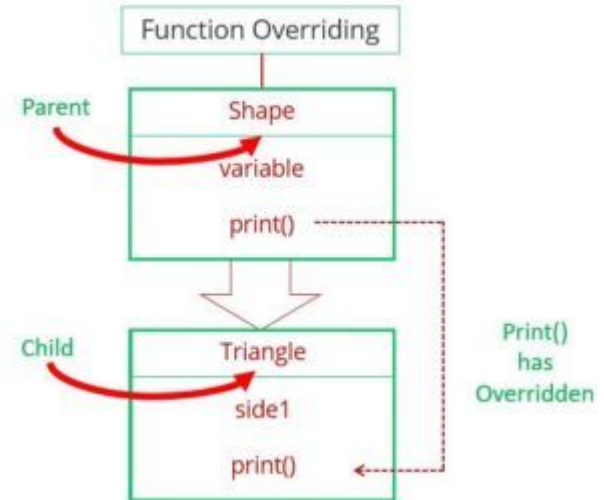


FUNCTION OVERRIDING

When overriding a method, the behavior of the method is changed for the derived class.

A class may need to override the default behavior provided by its base class.

- Provide behavior specific to a derived class
- Extend the default behavior
- Restrict the default behavior
- Improve performance



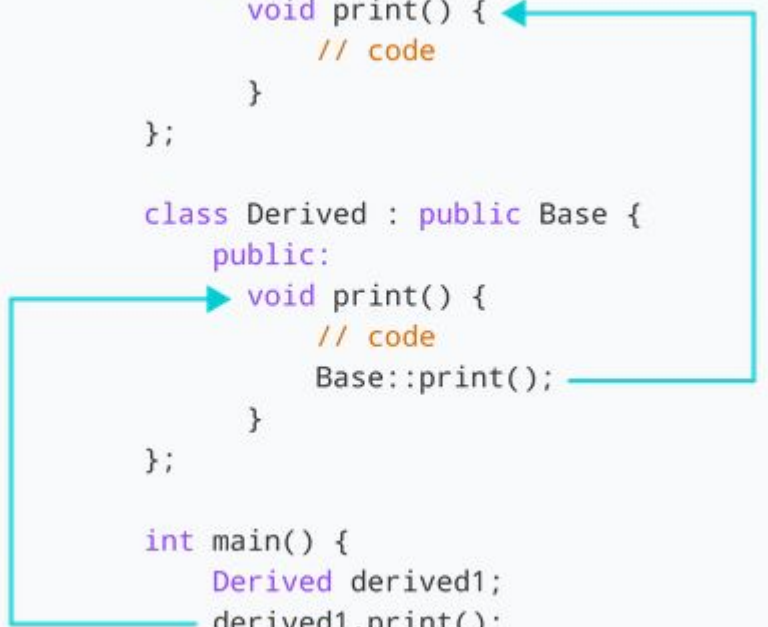
FUNCTION OVERRIDING

```
class Base {  
    public:  
    void print() {  
        // code  
    }  
};  
  
class Derived : public Base {  
    public:  
    void print() {  
        // code  
    }  
};  
  
int main() {  
    Derived derived1, derived2;  
  
    derived1.print();  
  
    derived2.Base::print();  
  
    return 0;  
}
```

The diagram illustrates function calls from the `main()` function. A blue line originates from `derived1.print();` and points to the `print()` method of the `Derived` class. Another blue line originates from `derived2.Base::print();` and points to the `print()` method of the `Base` class.

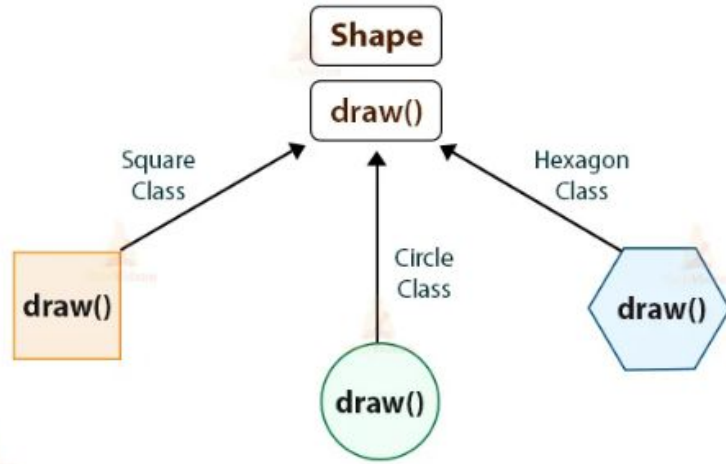
FUNCTION OVERRIDING

```
class Base {  
    public:  
    void print() {  
        // code  
    }  
};  
  
class Derived : public Base {  
    public:  
    void print() {  
        // code  
        Base::print();  
    }  
};  
  
int main() {  
    Derived derived1;  
    derived1.print();  
    return 0;  
}
```



FUNCTION OVERRIDING - SPECIFIC BEHAVIOUR

When overriding a method, the behavior of the method is changed for the derived class.



When I override my parent's methods



FUNCTION OVERRIDING - IMPROVE PERFORMANCE

When overriding a method, the behavior of the method is changed for the derived class.

FUNCTION OVERRIDING

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

class base_class
{
public:
    void findArea(int side)
    {cout << "Base::The area is: "<<(side * side) << "\n\n"; }
    void findArea(int x, int y)
    {cout << "Base::The area is: "<<(x * y) << "\n\n"; }
};

class derived_class : public base_class
{
public:
    void findArea(int side)
    { cout << "Derived::The area is: "<<(side * side) << "\n\n"; }
    void findArea(int x, int y)
    {cout << "Derived::The area is: "<<(x * y) << "\n\n"; }
};
```

```
int main()
{
    derived_class obj;
    obj.findArea(5);
    obj.findArea(5,4);
    obj.base_class::findArea(6);
};

return 0;

}
```

Make a function `attend_meeting()` in `Programmer` class. (`print I am attending as a Programmer`)

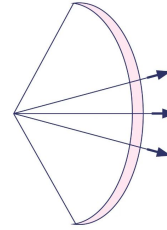
Both `Developer` and `Tester` will override the function called `attend_meeting()` that allows them to attend team meetings and provide updates on their progress and any issues they may be facing.

(`print I am attending as a Developer/Tester`)

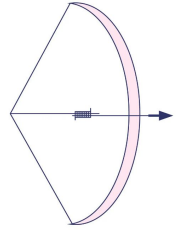
However, Developer has an overloaded `attend_meeting()` function that takes in a presentation parameter and presents their work to the team, while Tester has an overloaded `attend_meeting()` function that takes in a question parameter and answers any questions related to testing or quality assurance.

OVERLOADING VS OVERRIDING

Overloading



Overriding



Method Overloading	Method Overriding
In Method overloading methods must have different signature.	In method overriding methods must have same signature.
Function Overloading is to “add” or “extend” more to method’s behavior.	Function overriding is to completely “change” or “redefine” the behavior of a method.
Method overloading is used to achieve Compile time polymorphism	Method overriding is used to achieve run time polymorphism.
In method/function overloading, compiler knows which object assigned to which class at the time of compilation.	In method overriding, which object assigned to which class is not known till runtime.
Function Overloading takes place in a same class.	Overriding takes place in a class derived from a base class.