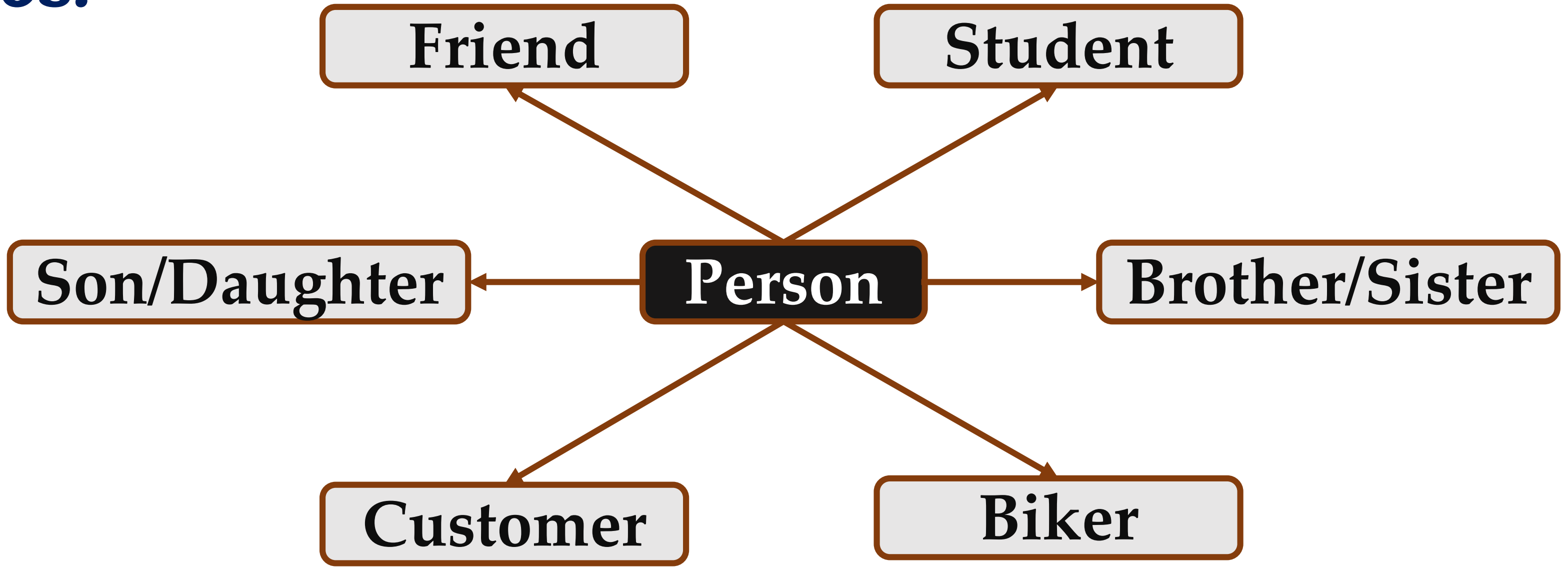
Polymorphism in OOP with C++

Polymorphism

- The word ***polymorphism (Poly + Morphism)*** means having ***many forms***. In simple words, we can define polymorphism as the ability of something to be displayed in more than one form or ***the same entity (function or object) behaves differently in different scenarios.***

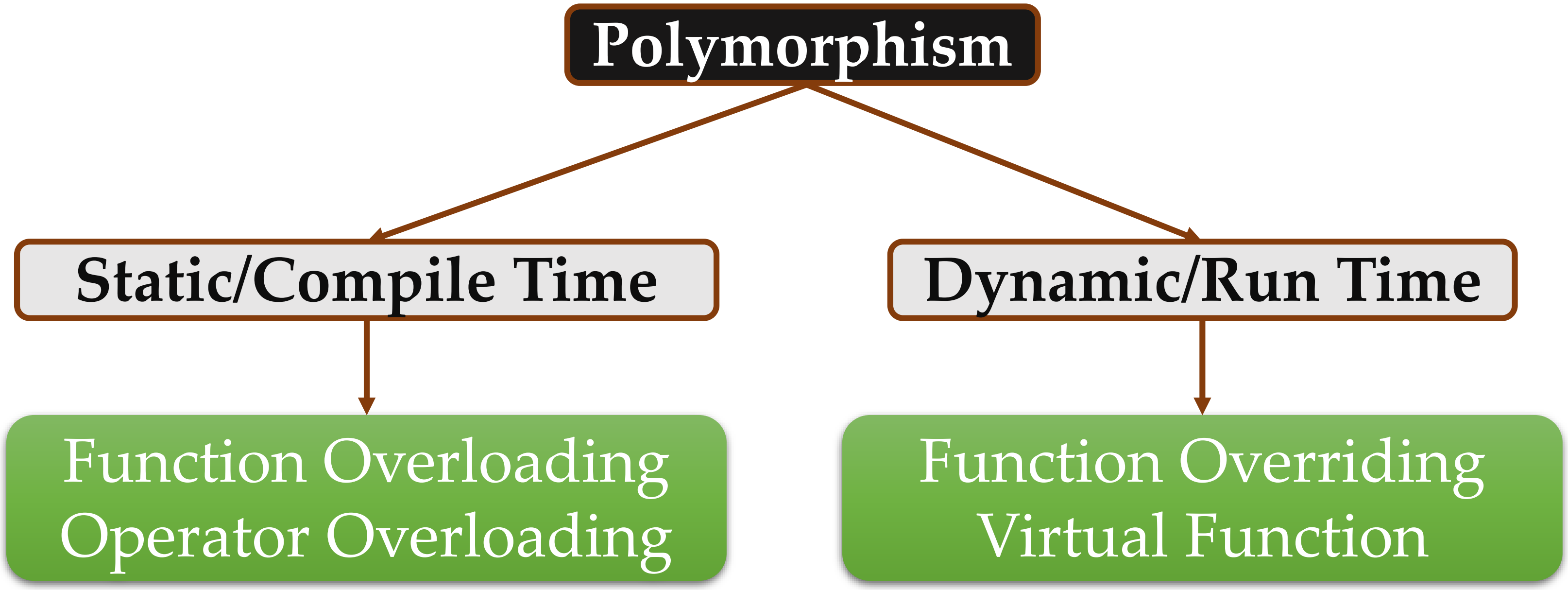


An Example of Polymorphism

- The **“+”** operator in C++ can perform two specific functions at two different scenarios i.e when the “+” operator is used in numbers, it performs ***addition***.
- ***int a = 10, b= 20; int sum = a + b; //sum = 30***

- And the same “+” operator is used in the string, it performs ***concatenation***.
- ***string s1 = “University”, s2= “Students”;***
- ***string s = s1 + s2; //s = University Students***

Types of Polymorphism



Compile Time Polymorphism

- This type of polymorphism is achieved by function overloading or operator overloading.
- **Function Overloading**: When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

Function Overloading

- ***Function overloading*** is a feature of object-oriented programming where two or more functions can have the ***same name but different parameters.***
- The function overloading feature is used to improve the ***readability of the code.***
- It is used so that the programmer does not ***have to remember various function names.***

Rules of Function Overloading

- ***Functions have different parameter type*** like,

  sum(int a, int b); sum(double a, double b).

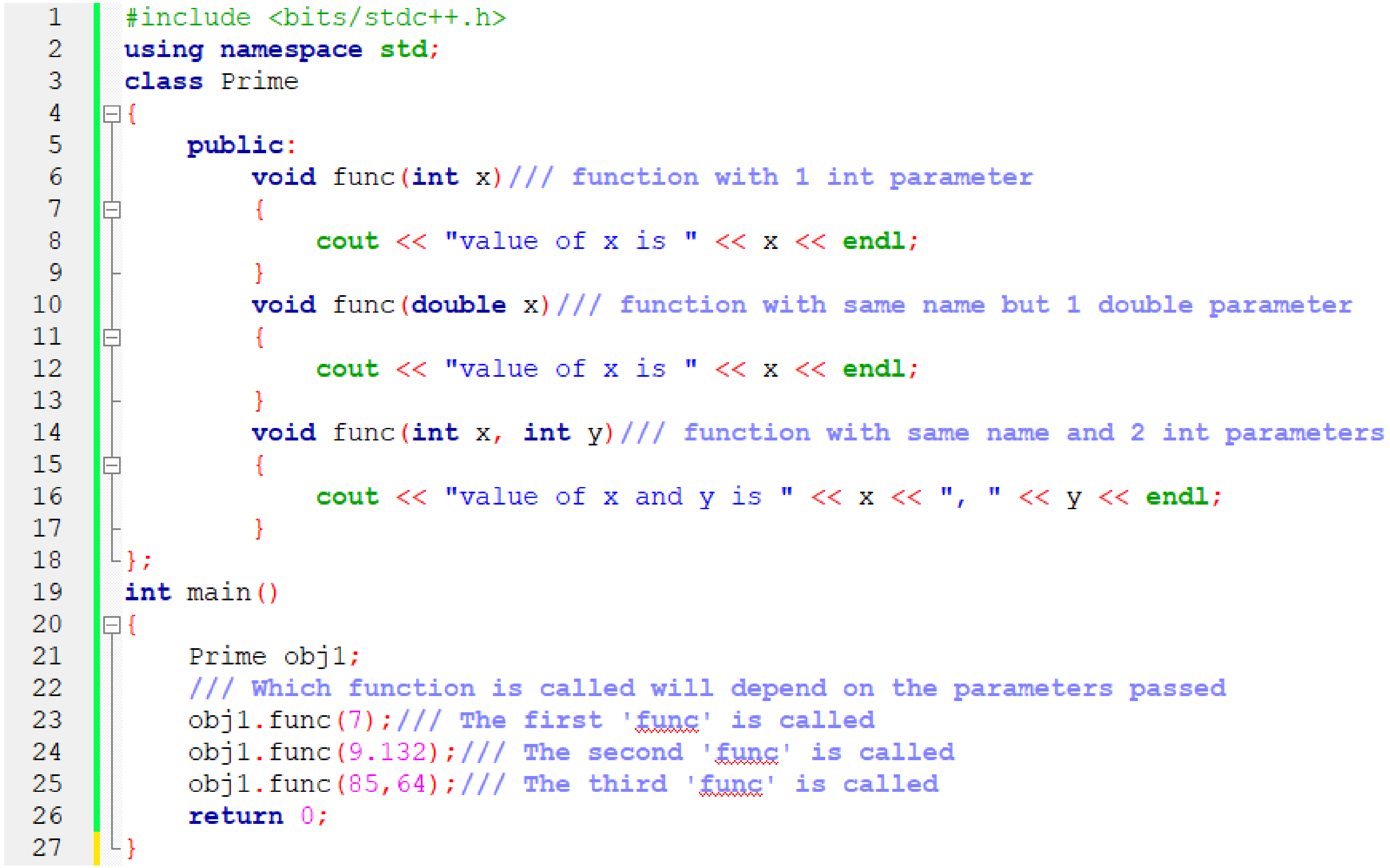
- ***Functions have a different number of parameters*** like,

  sum(int a, int b); sum(int a, int b, int c)

- ***Functions have a different sequence of parameters*** like,

  sum(int a, double b); sum(double a, int b).

Compile Time Polymorphism(Function Overloading)



```cpp
#include <bits/stdc++.h>
using namespace std;
class Prime
{
    public:
        void func(int x)/// function with 1 int parameter
        {
            cout << "value of x is " << x << endl;
        }
        void func(double x)/// function with same name but 1 double parameter
        {
            cout << "value of x is " << x << endl;
        }
        void func(int x, int y)/// function with same name and 2 int parameters
        {
            cout << "value of x and y is " << x << ", " << y << endl;
        }
};
int main()
{
    Prime obj1;
    /// Which function is called will depend on the parameters passed
    obj1.func(7);/// The first 'func' is called
    obj1.func(9.132);/// The second 'func' is called
    obj1.func(85,64);/// The third 'func' is called
    return 0;
}
```

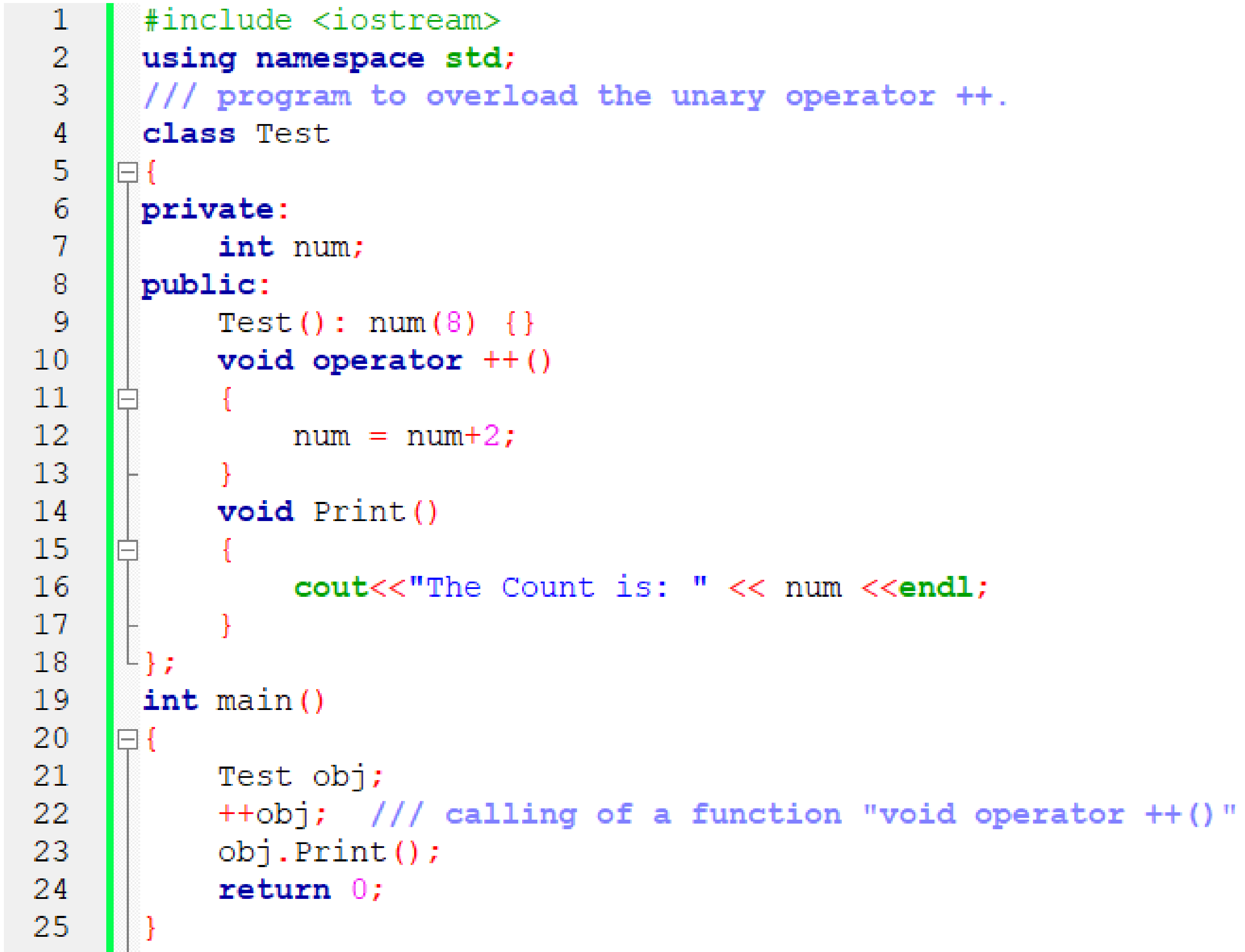
Compile Time Polymorphism

- ➢**Operator Overloading**: C++ also provide option to overload operators.
- ✓For example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands.
- ✓So a single operator '+' when placed between integer operands, adds them and when placed between string operands, concatenates them.
- ✓The advantage of Operators Overloading is to perform different operations on the same operand.

Rules of Operator Overloading

- Existing operators can only be overloaded, but the new operators cannot be overloaded.
- The overloaded operator contains at least one operand of the user-defined data type.
- We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.
- When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
- When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

Compile Time Polymorphism(Operator Overloading)

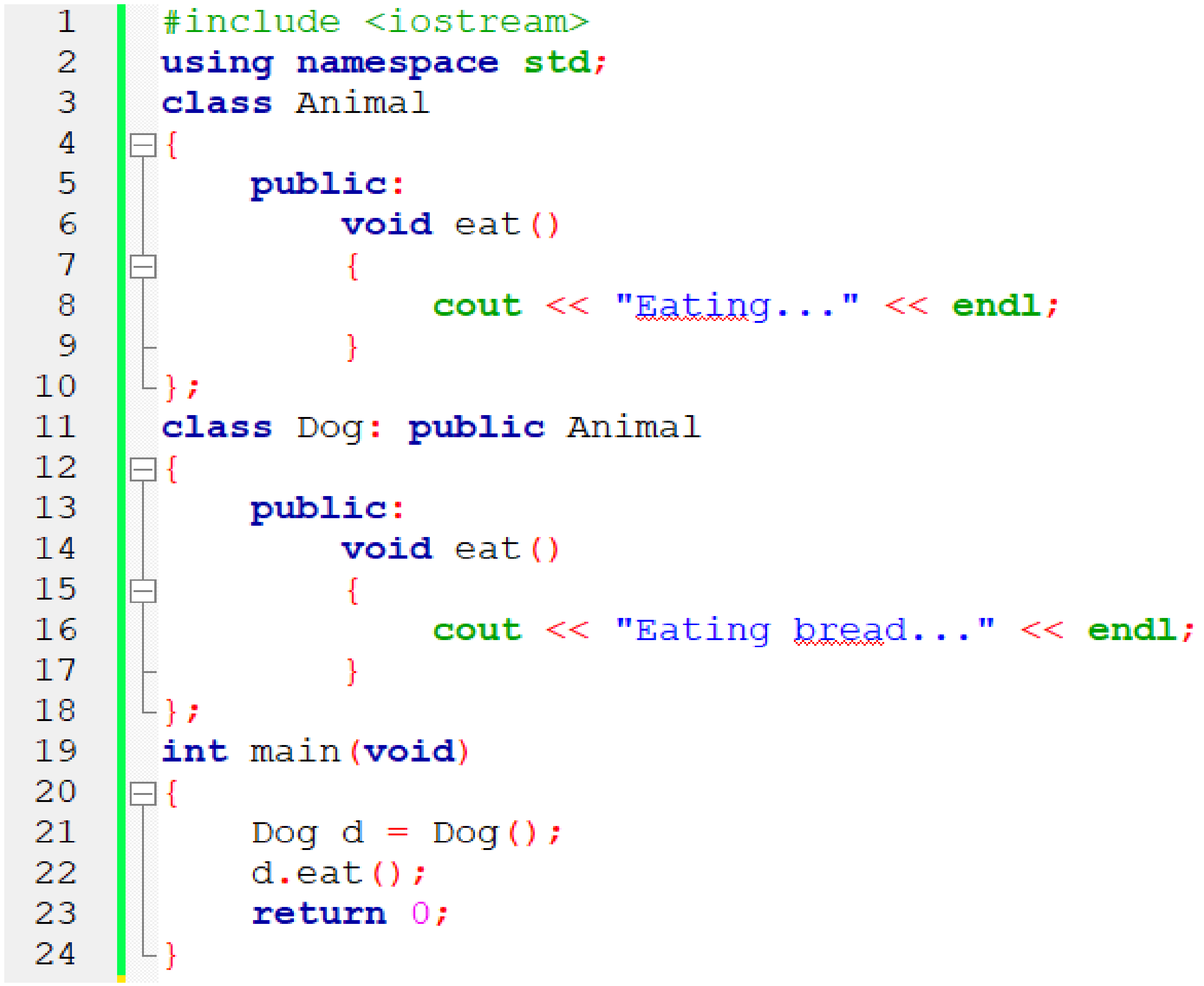


```cpp
#include <iostream>
using namespace std;
/// program to overload the unary operator ++.
class Test
{
private:
    int num;
public:
    Test(): num(8) {}
    void operator ++()
    {
        num = num+2;
    }
    void Print()
    {
        cout<<"The Count is: " << num <<endl;
    }
};
int main()
{
    Test obj;
    ++obj;  /// calling of a function "void operator ++()"
    obj.Print();
    return 0;
}
```

Run Time Polymorphism

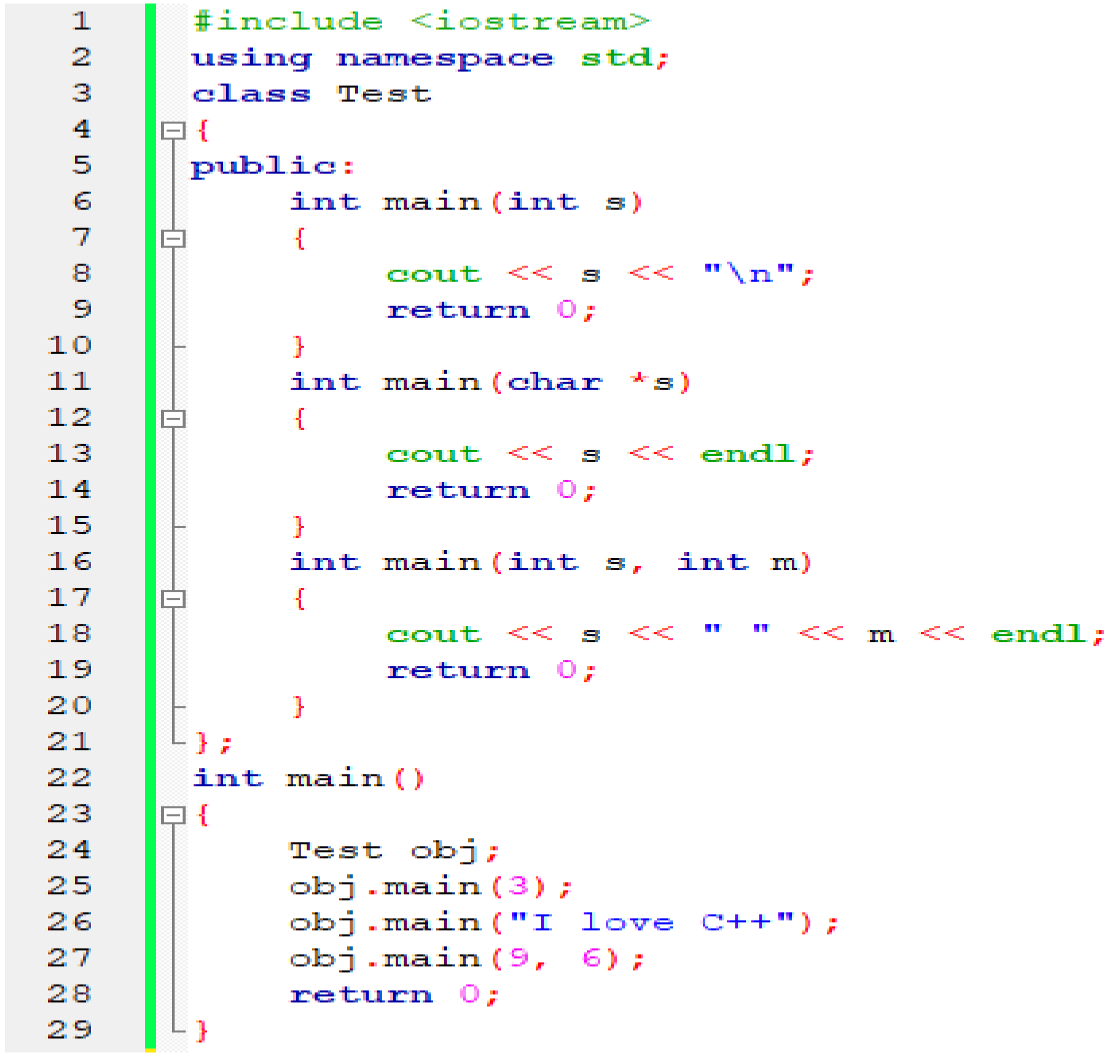
- This type of polymorphism is achieved by Function Overriding.
- **Function overriding** on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

Run Time Polymorphism(Overriding)



```
#include <iostream>
using namespace std;
class Animal
{
    public:
        void eat()
        {
            cout << "Eating..." << endl;
        }
};
class Dog: public Animal
{
    public:
        void eat()
        {
            cout << "Eating bread..." << endl;
        }
};
int main(void)
{
    Dog d = Dog();
    d.eat();
    return 0;
}
```

Overloading main() function in C++



```cpp
#include <iostream>
using namespace std;
class Test
{
public:
        int main(int s)
        {
                cout << s << "\n";
                return 0;
        }
        int main(char *s)
        {
                cout << s << endl;
                return 0;
        }
        int main(int s, int m)
        {
                cout << s << " " << m << endl;
                return 0;
        }
};
int main()
{
        Test obj;
        obj.main(3);
        obj.main("I love C++");
        obj.main(9, 6);
        return 0;
}
```

Function Overloading VS Function Overriding in C++

- **Inheritance:** Overriding of functions occurs when one class is inherited from another class. Overloading can occur without inheritance.
- **Function Signature:** Overloaded functions must differ in function signature i.e. either number of parameters or type of parameters should differ. In overriding, function signatures must be same.
- **Scope of functions:** Overridden functions are in different scopes; whereas overloaded functions are in same scope.
- **Behavior of functions:** Overriding is needed when derived class function has to do some added or different job than the base class function. Overloading is used to have same name functions which behave differently depending upon parameters passed to them.

Thank You