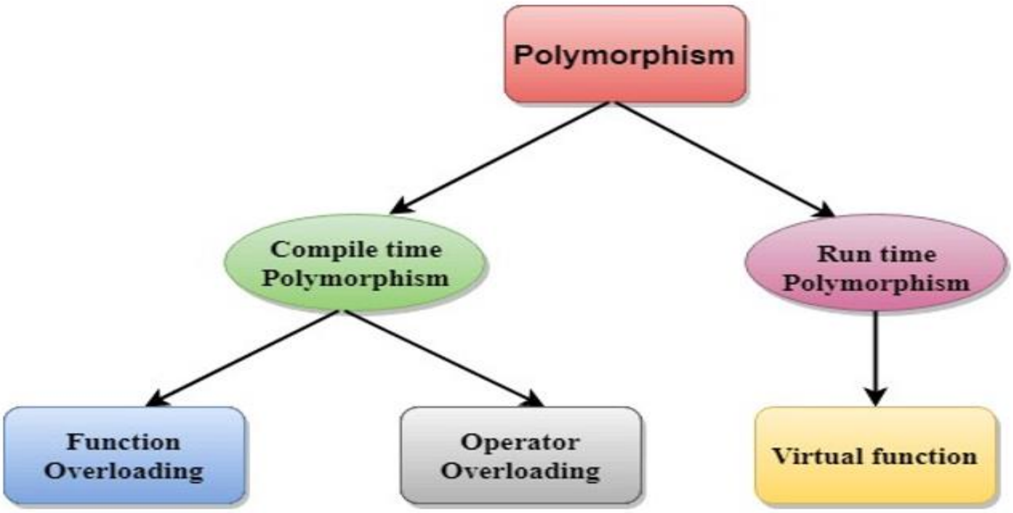
Polymorphism

- Polymorphism is a concept that allows you to perform a single action in different ways.
- Polymorphism is the combination of two Greek words.
- The poly means many, and morphs means forms.
- So polymorphism means many forms.
- Real-life example: A person at the same time can have different characteristics.
- Like aman at the same time is a father, a husband, and an employee.
- So the same person possesses different behavior in different situations.

Two types of polymorphism in C++



Compile Time Polymorphism:

- It is also known as static polymorphism.
- This type of polymorphism can be achieved through function overloading or operator overloading.

Function overloading:

- When there are multiple functions in a class with the same name but different parameters, these functions are overloaded.
- The main advantage of function overloading is that it increases the program's readability.
- Functions can be overloadedby using different numbers of arguments or by using different types of arguments.

Operator Overloading:

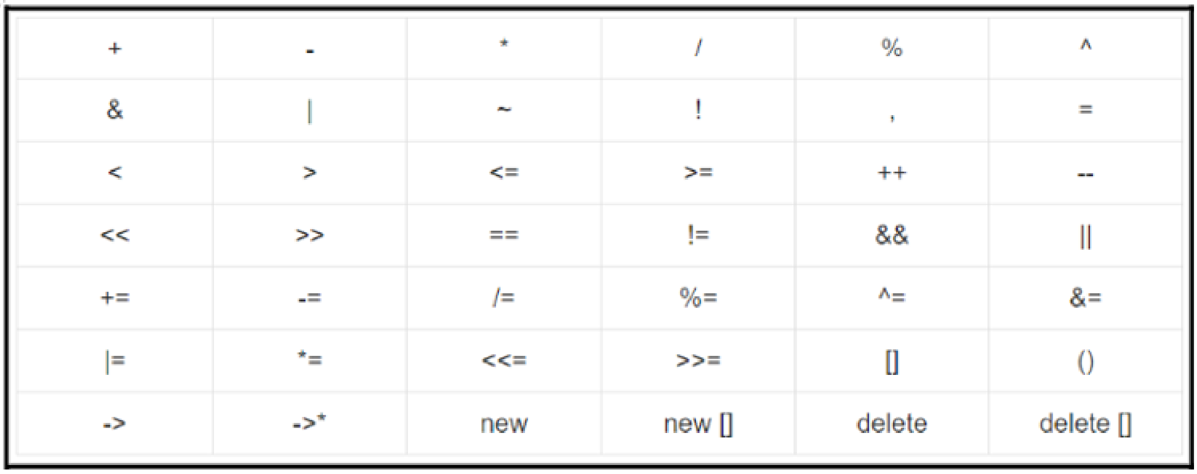
- C++ also provides options to overload operators.
- For example, we can make the operator ('+') for the string class to concatenate two strings.
- We know that this is theaddition operator whose task is to add two operands.
- When placed between integer operands, a single operator, '+,' adds them and concatenates them when placed between string operands.

Points to remember while overloading an operator:

- It can be used only for user-defined operators(objects, structures) but cannot be used for in-built operators(int, char, float, etc.).

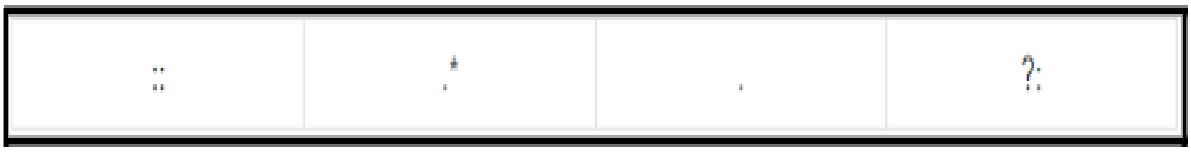
- Operators = and & are already overloaded in C++ to avoid overloading them.
- The precedence and associativity of operators remain intact.

List of operators that can be overloaded in C++:

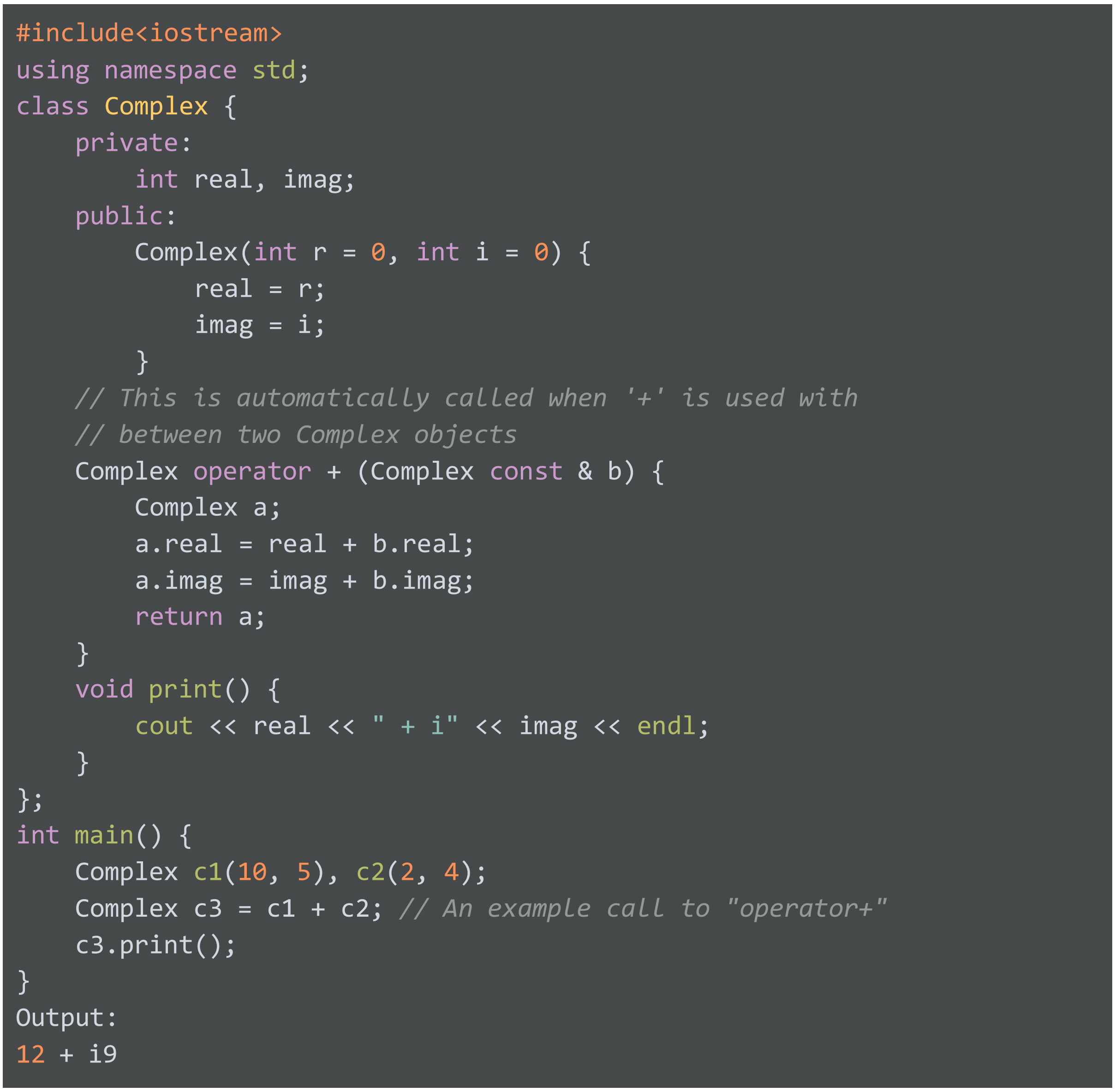


+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

List of operators that cannot be overloaded in C++:



::	.*	.	?:



```
#include<iostream>
using namespace std;
class Complex {
    private:
        int real, imag;
    public:
        Complex(int r = 0, int i = 0) {
            real = r;
            imag = i;
        }
    // This is automatically called when '+' is used with
    // between two Complex objects
    Complex operator + (Complex const & b) {
        Complex a;
        a.real = real + b.real;
        a.imag = imag + b.imag;
        return a;
    }
    void print() {
        cout << real << " + i" << imag << endl;
    }
};
int main() {
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
}
Output:
12 + i9
```

Example: Perform the addition of two imaginary or complex numbers.

Runtime polymorphism:

- It is also known as dynamic polymorphism.
- Method overriding is a way to implement runtime polymorphism.

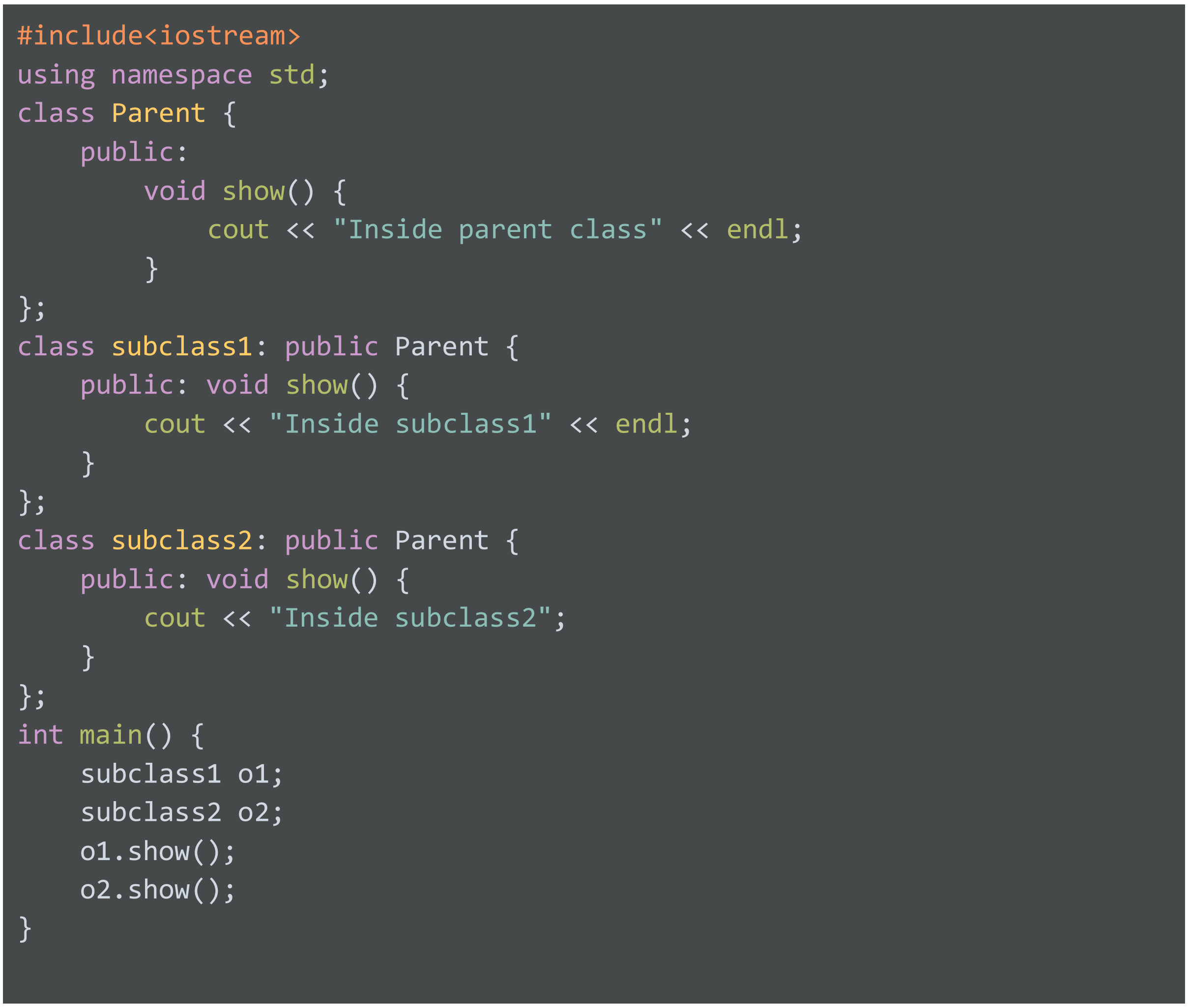
Method overriding:

- Method overriding is a feature that allows you to redefine the parent class method in the child class based on its requirement.
- In other words, whatever methods the parent class has by default are available in the child class.
- But, sometimes, a child class may notbe satisfied with parent class method implementation.
- The child class is allowed to redefine that method based on its requirement. This process is called method overriding.

Rules for method overriding:

- The parent class method and the method of the child class must have the samename.
- The parent class method and the method of the child class must have the sameparameters.
- It is possible through inheritance only.

Example:



```cpp
#include<iostream>
using namespace std;
class Parent {
    public:
        void show() {
            cout << "Inside parent class" << endl;
        }
};
class subclass1: public Parent {
    public: void show() {
        cout << "Inside subclass1" << endl;
    }
};
class subclass2: public Parent {
    public: void show() {
        cout << "Inside subclass2";
    }
};
int main() {
    subclass1 o1;
    subclass2 o2;
    o1.show();
    o2.show();
}
```

```
Output:
Inside subclass1
Inside subclass2
```