C++ Runtime polymorphism example : By using two derived classes

Below is an example with virtual keyword.

eg.

```
class Shape {                                   // base class
    public:
    virtual void draw(){
            cout << "drawing." << endl;         // virtual function
        }
    };

class Rectangle : public shape {                // inheriting shape class
    public:
    void draw() {
        cout << "drawing rectangle..." << endl;
      }
    };

class Circle : public Shape {
     public:
       void draw(){
       cout << "drawing circle..." << endl;
       }
    };

 int main() {
     shape * s ;                     // base class pointer
     shape   sh ;                    // base class object
Rectangle rec;
Circle cir;

s = &sh;
s -> draw();
s = &rec
s -> draw();
s = &cir
s -> draw();
}
```

output

```
drawing...
drawing rectangle...
drawing circle...
```

Runtime Polymorphism with Data Members

Runtime polymorphism can be achieved by data members in C++. Below is an example where we access the field by reference variable which refers to the instance of derived class.

eg:

```
class Animal {
    public:
      string color = "Black";
};
class Dog : public Animal {
    public:
       string color = 'grey';
};
```

```
int main () {

    Animal d = Dog() ;          // ??

    cout << d.color ;

}
```

Output

```
Black.
```

C++ Overloading (Function and operator)

If we create two or more members having the same name but different in number or type of parameters, it is known as C++ overloading. In C++, we can overload:

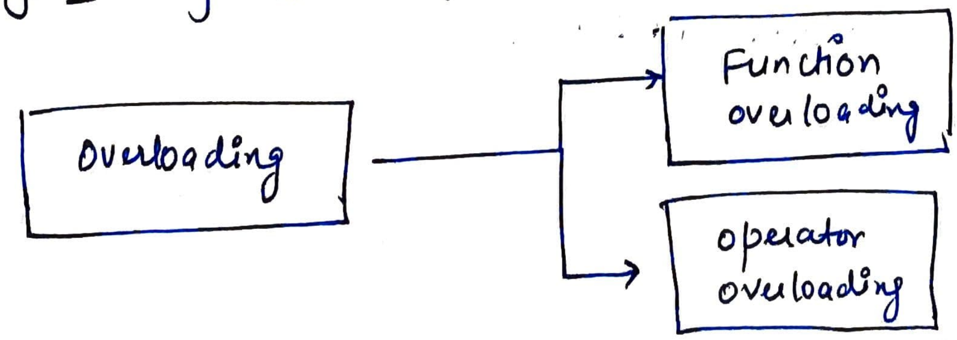
(i) methods

(ii) constructors

(iii) indexed properties

It is because these members have parameters.

⇒ Types of overloading in C++



⇒ C++ Function overloading

Function overloading is defined as the process of having two or more function with the same name, but different parameters.

A function is redefined by using either different types of arguments or a different number of arguments.
It is only through these differences compiler can differentiate between the functions.

Advantage of function overloading is that it increases the readability of the program. because you don't need different function names for same action.

eg: Number of arguments vary.

```
class Cal {
    public:
    static int add (inta, int b){
           return a + b;
    }
    static int add ( int a, int b, int c){
           return a + b + c;
      }
};
```

```
int main() {

    cal c;

    cout <<  c.add(10,20) << endl;

    cout <<  c.add(12,20,23) << endl;

}
```

output

```
    30
    55
```

→ eg. overloading with diff types of arguments

```
int mul (inta, int b) {

    return a x b;

}

float mul ( double x, inty ) {

    return x x y;

}

int main () {

    int r1  = mul (6,7);

    float r2 =  mul (0.2,3);

    cout << " r1 "<< r1 << endl;

    cout << " r2 " << r2 << endl;

}
```

output

```
r1   42

r2   0.6
```