



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Polymorphism

Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales

www.uwtsd.ac.uk



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022



Recap

<https://forms.office.com/e/HQVbhD1uek>

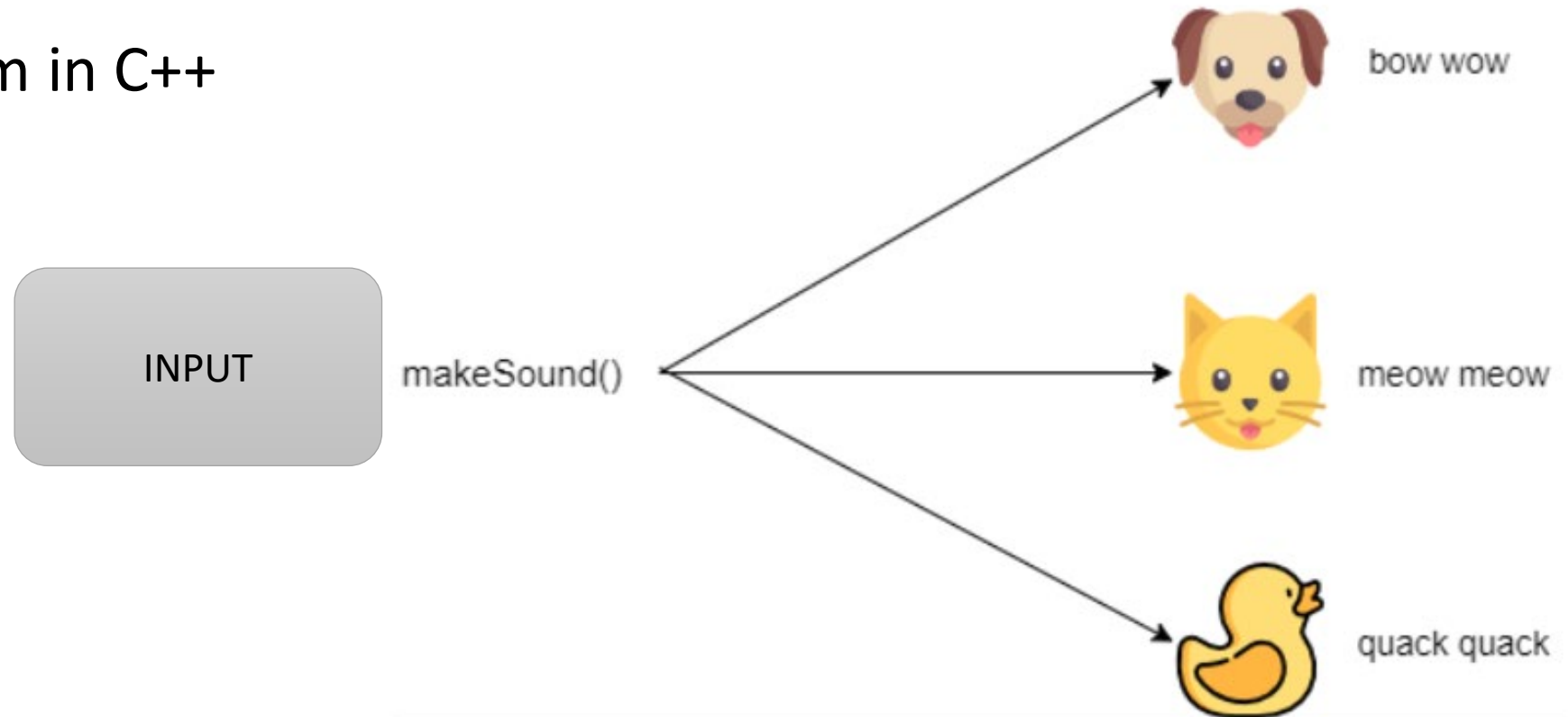
Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales

www.uwtsd.ac.uk



Objective:

- Polymorphism in C++



Introduction to Polymorphism

- Polymorphism means "many forms"
- Enables a function to take different forms based on the object it is acting upon.
- Implementation of polymorphism in C++
 - function overloading,
 - operator overloading,
 - function overriding
 - and runtime polymorphism using virtual functions.

Why Polymorphism?

- Polymorphism allows us to create consistent code.

For example,

Suppose we need to calculate the area of a circle and a square. To do so, we can create a Shape class and derive two classes Circle and Square from it.

- In this case, create a function having the same name calculateArea() in both the derived classes rather than creating functions with different names, thus making our code more consistent.

- **Operator Overloading:**

Redefining operators to work with user-defined types.

- Example:

Suppose we have created three objects **C1,C2** and **result** from a class named **Complex**

result = c1 + c2;

instead of **result=c1.addnumbers(c2);**

• Syntax for C++ Operator Overloading

```
class className {  
    . . . . .  
    public  
        returnType operator symbol (arguments)  
    {  
        . . . . .  
    }  
    . . . . .  
};
```

Here,

- returnType is the return type of the function.
- operator is a keyword.
- symbol is the operator we want to overload. Like: +, <, -, ++, etc.
- arguments is the arguments passed to the function.

Operator Overloading in Unary Operators

// Overload ++ when used as prefix

```
#include <iostream>
using namespace std;
class Count {
private:
    int value;
public:
    // Constructor to initialize count to 5
    Count() : value(5) {}
    // Overload ++ when used as prefix
    void operator ++ () {
        ++value;
    }
}
```

```
void display() {
    cout << "Count: " << value << endl;
}
};
int main() {
    Count count1;

    // Call the "void operator ++ ()" function
    ++count1;

    count1.display();
    return 0;
}
```

```
// Overload ++ when used as postfix  
void operator ++ (int) {  
    value++;  
}
```

```
// Call the "void operator ++ (int)" function  
count1++;  
count1.display();
```

Operator Overloading in Binary Operators

- When we overload the binary operator for user-defined types by using the code:

`obj3 = obj1 + obj2;`

- The operator function is called using the `obj1` object and `obj2` is passed as an argument to the function.

C++ program to add two complex numbers (overload the binary operator +)

```
class Complex {
    private:
        float real;
        float imag;

    public:
        Complex() : real(0), imag(0){}

        void input() {
            cout << "Enter real & imaginary
            parts respectively: ";
            cin >> real;
            cin >> imag;
        }
}
```

```
// Overload the + operator
Complex operator + (const Complex& obj) {
    Complex temp;
    temp.real = real + obj.real;
    temp.imag = imag + obj.imag;
    return temp;
}

void output() {
    if (imag < 0)
        cout << "Output Complex number: " << real << imag << "i";
    else
        cout << "Output Complex number: " << real << "+" << imag << "i";
    }
};
```



C++ program to add two complex numbers (overload the binary operator +)

Enter first complex number:
Enter real and imaginary parts
respectively: 9 5
Enter second complex
number:
Enter real and imaginary parts
respectively: 7 6
Output Complex number:
16+11i

```
int main() {  
    Complex complex1, complex2, result;  
  
    cout << "Enter first complex number:\n";  
    complex1.input();  
  
    cout << "Enter second complex number:\n";  
    complex2.input();  
  
    // complex1 calls the operator function  
    // complex2 is passed as an argument to the function  
    result = complex1 + complex2;  
    result.output();  
  
    return 0;  
}
```

There are 4 operators that cannot be overloaded in C++. They are:

:: → (scope resolution)

. → (member selection)

.* → (member selection through pointer to function)

?: → (ternary operator)

C++ Function Overriding

```
class Base {  
    public:  
    virtual void print() {  
        cout << "Base Function" << endl;  
    }  
};
```

```
class Derived : public Base {  
    public:  
    void print() {  
        cout << "Derived Function" << endl;  
    }  
};
```

```
int main() {  
    Derived derived1;  
  
    // Call print() function of Derived class  
    derived1.print();  
  
    return 0;  
}
```

C++ Virtual Functions

- Using virtual functions in the base class ensures that the function can be overridden in these cases.
- In C++, we may not be able to override functions if we use a pointer of the base class to point to an object of the derived class.

// C++ program to demonstrate the use of virtual functions

```
int main()
{
    Derived derived1;

    // pointer of Base type that points to derived1
    Base* base1 = &derived1;

    // calls member function of Derived class
    base1->print();

    return 0;
}
```

```
#include <iostream>
using namespace std;
class Base {
public:
    virtual void print() {
        cout << "Base Function" << endl;
    };
};

class Derived : public Base {
public:
    void print() {
        cout << "Derived Function" << endl;
    }
};
```

End of Session

----next session
Encapsulation



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales