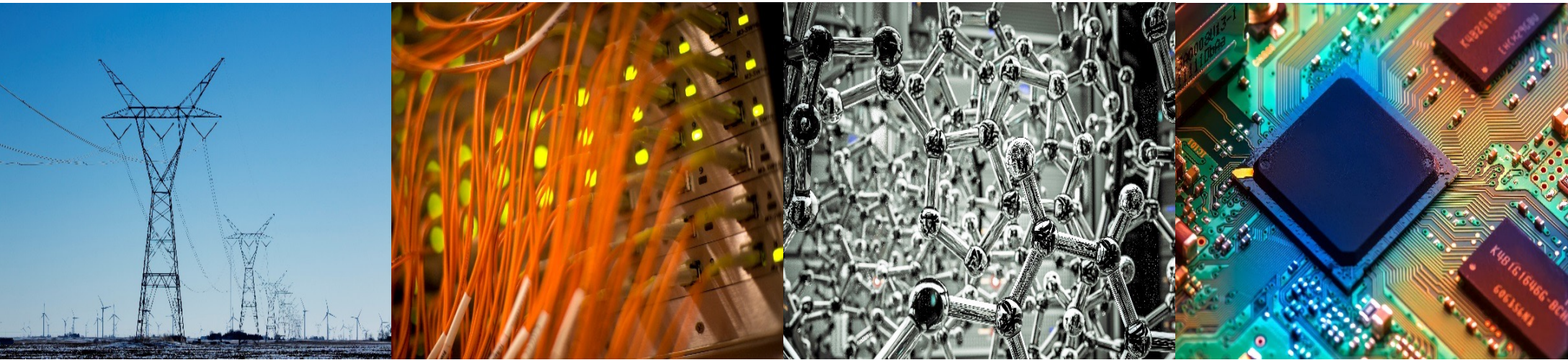


ECE 220 Computer Systems & Programming

Lecture 21 – Introduction to C++: Inheritance & Polymorphism

April 14, 2026



- Quiz4 is this week
- MT2 regrade request is due on 4/15

I ILLINOIS

Electrical & Computer Engineering

GRAINGER COLLEGE OF ENGINEERING

Pass by Value / Address (Pointer) / Reference in C++

Let's look at our most familiar *swap* example.

1. Pass by value

```
void swap_val(int x, int y);
```


2. Pass by address (pointer)

```
void swap_ptr(int *x, int *y);
```


3. Pass by reference

```
void swap_ref(int &x, int &y);
```

```
int main(){
    int a = 1;
    int b = 2;
    swap_val(a, b);    //pass by value
    swap_ptr(&a, &b);  //pass by address (pointer)
    swap_ref(a, b);    //pass by reference
    return 0;
}
```



```
void swap_ptr(int *x, int *y){
    int temp = *x;
    *x = *y;
    *y = temp;
}
```



```
void swap_ref(int &x, int &y){
    int temp = x;
    x = y;
    y = temp;
}
```

More on C++ Reference

- an alias for a variable/object
- similar to pointer but safer
- no need to dereference, use it just like a variable/object
- should use “.” instead of “->” to access members

Copy constructor and **pass by constant reference**

```
class vector{
    Protected:
    double angle_, length_;
    public:
    //copy constructor
    vector(const vector &obj) {
        angle_ = obj.angle_;
        length_ = obj.length_;}
    //other methods omitted here for simplicity
};
```

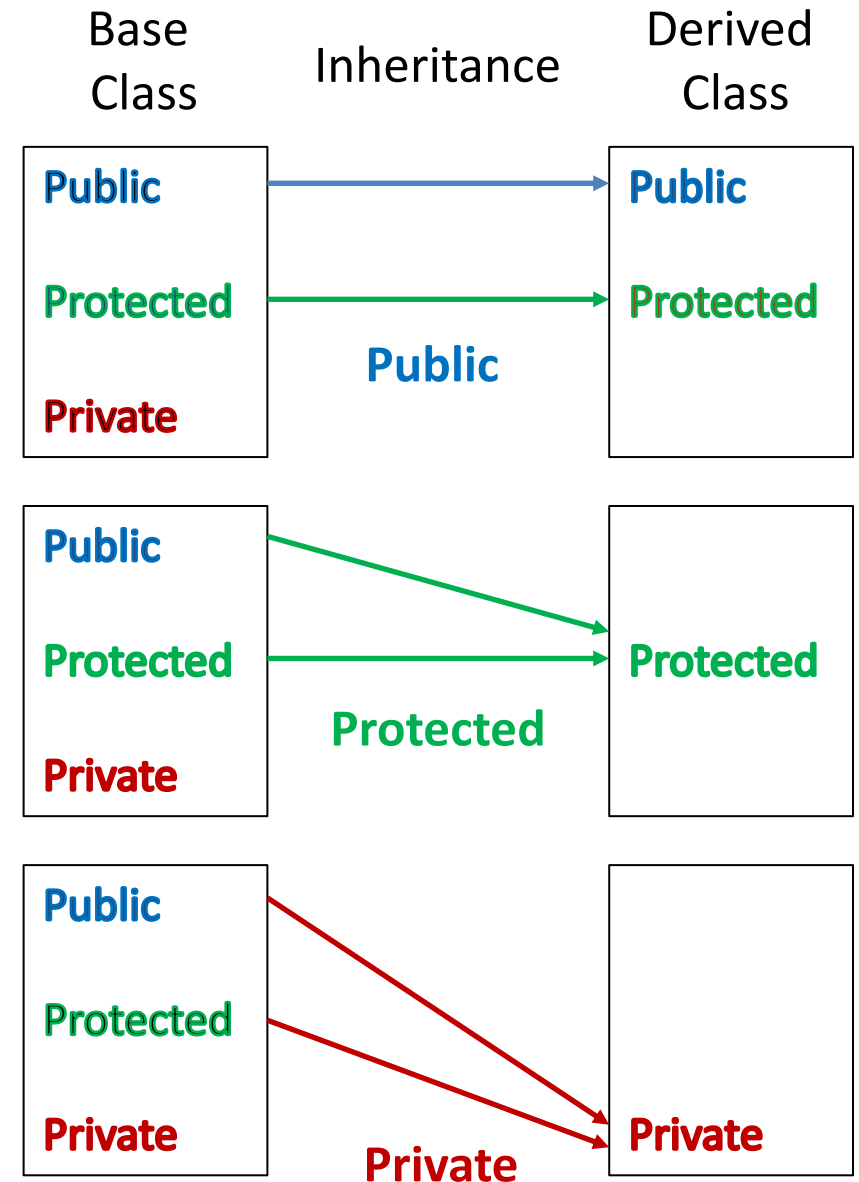
Inheritance

C++ allows us to define a class based on an existing class, and the new class will inherit members of the existing class.

- the **existing** class –
- the **new** class –

Exceptions in inheritance (things not inherited):

- constructors, destructors and copy constructors of the base class
 - overloaded operators of the base class
 - friend functions of the base class
- Are private members in the base class inherited?



```

class orthovector : public vector{
    protected:
    int d_; //direction can be 0,1,2,3, indicating r, l, u, d
    public:
    orthovector(int dir, double l){
        const double halfPI = 1.507963268;
        d_ = dir;
        angle_ = dir*halfPI;
        length_ = l;
    }
    orthovector() {d_ = 0; angle_ = 0.0; length_ = 0.0;}
    double hypotenuse(const orthovector &b){
        if((d_+b.d_)%2 == 0) return length_ + b.length_;
        return (sqrt(length_*length_ + b.length_*b.length_));
    }
};

```

Access	public	protected	private
Same Class	Y	Y	Y
Derived Class	Y	Y	N
Outside Class	Y	N	N

Polymorphism

A call to a member function will cause a **different function to be executed** depending on the type of the object that invokes the function. In the example below, function call is determined during _____ (**static linkage**).

Example:

```
//base class
class Shape{
    protected:
    double width_, height_;
    public:
    Shape() {width_ = 0; height_ = 0;}
    Shape(double a, double b) { width_ = a; height_ = b; }
    double area() { cout << "Base class area unknown" << endl;
                    return 0; }
};
```

```
int main() {
    Rectangle rec(3,5);
    Triangle tri(4,5);

    rec.area();
    tri.area();

    return 0;
}
```

```
//derived classes
class Rectangle : public Shape{
    public:
    Rectangle(double a, double b) : Shape(a,b){}
    double area() {

    }

};
```

```
class Triangle : public Shape{
    public:
    Triangle(double a, double b) : Shape(a,b){}
    double area() {

    }

};
```

➤ Which function will be invoked when we execute the code in main?

Declared Type vs. Actual Type

```
int main() {  
    Shape *ptr;  
    Rectangle rec(3,5);  
    Triangle tri(4,5);  
  
    //use ptr to point to Rectangle class object  
    ptr = &rec;  
    ptr->area();  
  
    //use ptr to point to Triangle class object  
    ptr = &tri;  
    ptr->area();  
  
    return 0;  
}
```

➤ What would this program print?

Virtual Function

- member functions in the base class you expect to redefine in the derived classes are called **virtual functions**
- derived class declares instances of that member function
- function call is determined during _____ (**dynamic linkage**)

```
//base class
class Shape{
    protected:
    double width_, height_;
    public:
    Shape() {width_ = 0; height_ = 0;}
    Shape(double a, double b) { width_ = a; height_ = b; }
    virtual double area(){
        cout << "Base class area unknown" << endl;
        return 0; }
};
```

Virtual Function Table (vtbl)

- stores pointers to all virtual functions
- created for *each class* that uses virtual functions
- lookup during the function call

➤ Where are things being stored at?

Program Text (Code Segment)
Data (Static, Global, etc.)
Heap
Stack

Pure Virtual Functions & Abstract Class

```
class Shape{ //Shape is an abstract class
protected:
double width_, height_;
public:
Shape(double a, double b) { width_ = a; height_ = b; }
virtual double area()=0; //pure virtual function
};

int main(){
    Shape shape1(2,4);    // this will cause a compiler error!
    Shape *p_shape1;    // this is allowed
}
```

More on Abstract Class

- a class with one or more pure virtual functions is an abstract class, no objects of that abstract class can be created
- a pure virtual function that is not defined in a derived class remains a pure virtual function, so the derived class is also an abstract class
- an abstract class is intended as an **interface** to objects accessed through **pointers** and **references**