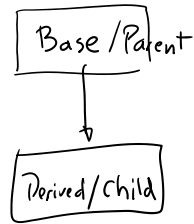


# ECE 244

## Programming Fundamentals

### Lec.25: Inheritance 4 Polymorphism

Ding Yuan  
ECE Dept., University of Toronto  
<http://www.eecg.toronto.edu/~yuan>



class child : class Parent {

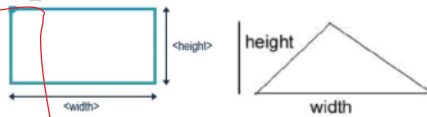
3;

class

### The problem

```

4 class Polygon {
5 protected:
6   int width, height;
7 public:
8   void set_values (int a, int b) { width=a; height=b; }
9 }; int area ( ) { return 0; }
  
```



```

10
11 class Rectangle: public Polygon{
12 public:
13   int area() {
14     return width*height;
15   }
16 class Triangle: public Polygon{
17 public:
18   int area() {
19     return width*height/2;
20   }
21 }
22 int main () {
23   Rectangle rect;
24   Triangle trgl;
25   Polygon * ppoly1 = &rect;
26   Polygon * ppoly2 = &trgl;
27   ppoly1->set_values (4, 5);
28   ppoly2->set_values (4, 5);
29   cout << rect.area() << '\n';
30   cout << trgl.area() << '\n';
31   return 0;
32 }
  
```

Handwritten notes and annotations:

- Red circle around line 9: `int area ( ) { return 0; }`
- Red circle around line 28: `cout << rect.area() << '\n';`
- Red circle around line 29: `cout << trgl.area() << '\n';`
- Red circle around line 31: `return 0;`
- Red circle around line 32: `}`
- Red circle around line 33: `But we can't do this: ppoly1->area();`
- Red circle around line 34: `:`
- Red circle around line 35: `:`
- Red circle around line 36: `:`
- Red circle around line 37: `:`
- Red circle around line 38: `:`
- Red circle around line 39: `:`
- Red circle around line 40: `:`
- Red circle around line 41: `:`
- Red circle around line 42: `:`
- Red circle around line 43: `:`
- Red circle around line 44: `:`
- Red circle around line 45: `:`
- Red circle around line 46: `:`
- Red circle around line 47: `:`
- Red circle around line 48: `:`
- Red circle around line 49: `:`
- Red circle around line 50: `:`
- Red circle around line 51: `:`
- Red circle around line 52: `:`
- Red circle around line 53: `:`
- Red circle around line 54: `:`
- Red circle around line 55: `:`
- Red circle around line 56: `:`
- Red circle around line 57: `:`
- Red circle around line 58: `:`
- Red circle around line 59: `:`
- Red circle around line 60: `:`
- Red circle around line 61: `:`
- Red circle around line 62: `:`
- Red circle around line 63: `:`
- Red circle around line 64: `:`
- Red circle around line 65: `:`
- Red circle around line 66: `:`
- Red circle around line 67: `:`
- Red circle around line 68: `:`
- Red circle around line 69: `:`
- Red circle around line 70: `:`
- Red circle around line 71: `:`
- Red circle around line 72: `:`
- Red circle around line 73: `:`
- Red circle around line 74: `:`
- Red circle around line 75: `:`
- Red circle around line 76: `:`
- Red circle around line 77: `:`
- Red circle around line 78: `:`
- Red circle around line 79: `:`
- Red circle around line 80: `:`
- Red circle around line 81: `:`
- Red circle around line 82: `:`
- Red circle around line 83: `:`
- Red circle around line 84: `:`
- Red circle around line 85: `:`
- Red circle around line 86: `:`
- Red circle around line 87: `:`
- Red circle around line 88: `:`
- Red circle around line 89: `:`
- Red circle around line 90: `:`
- Red circle around line 91: `:`
- Red circle around line 92: `:`
- Red circle around line 93: `:`
- Red circle around line 94: `:`
- Red circle around line 95: `:`
- Red circle around line 96: `:`
- Red circle around line 97: `:`
- Red circle around line 98: `:`
- Red circle around line 99: `:`
- Red circle around line 100: `:`

→ [ ] compiler error

## The problem (cont.)

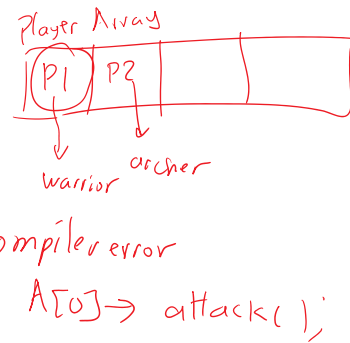
- What if you want to be able to invoke "area()" using a **pointer** to an object of base class

```

22 Rectangle rect;
23 Triangle trgl;
24 Polygon * ppoly1 = &rect;
25 Polygon * ppoly2 = &trgl;

```

- E.g., `ppoly1->area()` will invoke `rect.area()`, and `ppoly2->area()` will invoke `trgl.area()`
- More generally, we want use a pointer to the base class object to invoke a function that is implemented in a derived class
- You cannot do it using what we have learnt so far



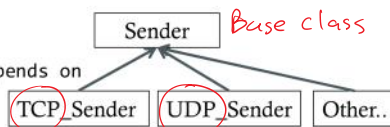
## Some real world examples

- A network system can use different protocols

```

void connect (Sender *sender) {
    sender->send (...);
    // Which send() is invoked depends on
    // the actual type of sender
}

```

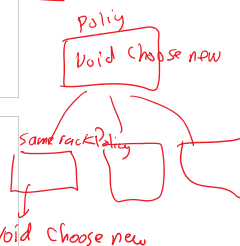
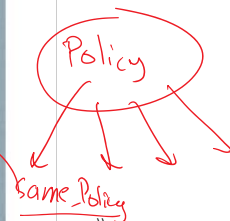
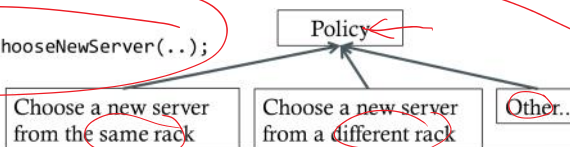


- Server fails frequently in data centers; you may have different policies to handle the failure

```

nv = policy->chooseNewServer(...);

```



## Data center



## Solution: virtual functions

If a function is declared as **virtual** in base class A, and redefined in derived class B, then a call made via a pointer, which is declared as A\* type but actually points to an obj. of type B, will cause the function defined in B being invoked

```
4 class Polygon {
5 protected:
6   int width, height;
7 public:
8   void set_values (int a, int b) { width=a; height=b; }
9   virtual int area() { return 0; }
10 };
11
12 class Rectangle: public Polygon {
13 public:
14   int area() { return width*height; }
15 };
16
17 class Triangle: public Polygon {
18 public:
19   int area() { return width*height/2; }
20 };
```

```
Rectangle rect;
Triangle trgl;
Polygon * ppoly1 = &rect;
Polygon * ppoly2 = &trgl;
ppoly1->set_values(4,5);
ppoly2->set_values(4,5);
cout<<ppoly1->area(); // 20
cout<<ppoly2->area(); // 10
```

What if we remove the "virtual" keyword from line 9?

Polygon poly;  
Polygon \* PP3 = &poly;  
PP3 -> area(); return 0;

⌚  
←  
←  
Polygon → Polygon::area(),  
Parent function ↑

function foo(Polygon\*) {

## Polymorphism

- A class that declares or inherits a virtual function is called a polymorphic class
  - Both Rectangle and Triangle are polymorphic classes of Polygon
  - Both TCP\_Sender and UDP\_Sender are polymorphic classes of Sender

## Virtual functions: more details

- The decision as to which inherited function to call is made at runtime when the function is invoked (rather than at compile time) depending on the type of the target object
  - This feature is also called late binding *runtime*
- The virtual function in Base class is like a "dummy" or "placeholder" declaration of the function
- You cannot declare a member variable as virtual
  - Only functions *virtual → operator =  
↳ destructor =*

*jump foo*

## Pure virtual functions

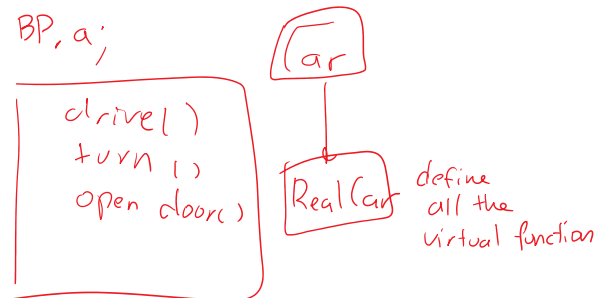
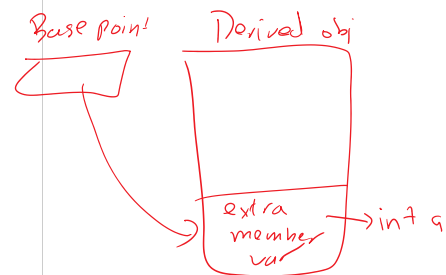
- If we never intended to use "Polygon" to create objects (i.e., we use the base only to derive classes), then we do not fill dummy function definition

```

4 class Polygon {
5 protected:
6   int width, height;
7 public:
8   void set_values (int a, int b) { width=a; height=b; }
9   virtual int area() = 0; // Pure virtual function
10 };
    
```

*BP, a;*  
*compile time*  
*extra member var*  
*int a*

- A class that contains at least one pure virtual function is called an abstract class
  - Also called an interface
- You cannot instantiate an object of an abstract class



## But there is no free lunch

- At runtime, a call to a virtual function is no longer a simple jump
- Needs to figure out the call target
  - Performance penalty: more expensive than a normal call
- Implementation: vtable (virtual method table)

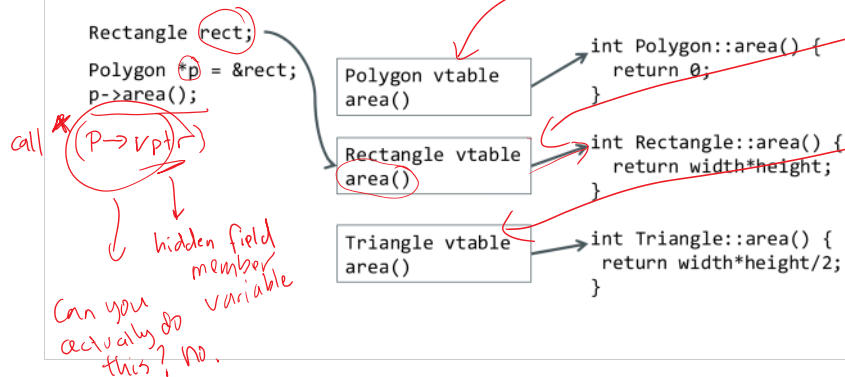


*abstract A virtual foo()=0;*  
↓  
*abstract B virtual foo()=0;*  
↓  
*real class C define foo();*

vptr

## vtable

- For every class, there is a table that maps each virtual function to the most-derived function target
- Each object has a pointer to the vtable of its type



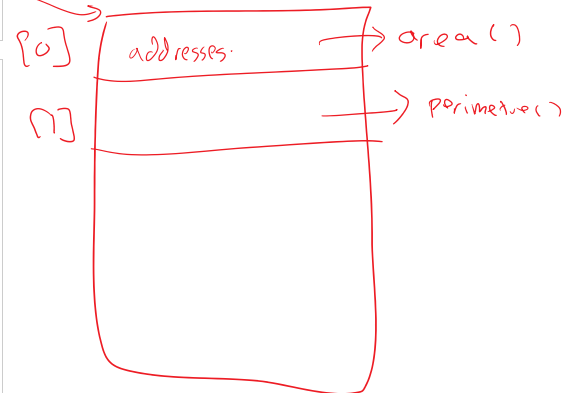
Polygon

field

Rectangle

Triangle

*vptr* → Func ~~no~~ *vptr*;  
Vtable



## Overhead

- On C++, this overhead is 6-13% compared w/software using only non-virtual function calls
  - Also prevent compilers from other optimizations, e.g., function inlining
- Modern systems use frameworks, tend to make things worse
- Just-In-Time compiler can use inline caching to reduce this overhead