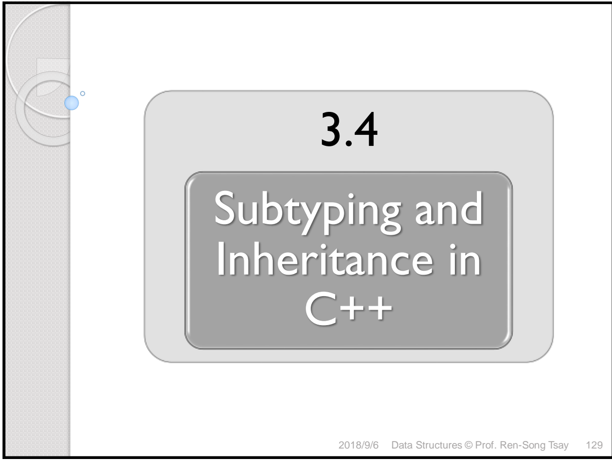


Object Inheritance

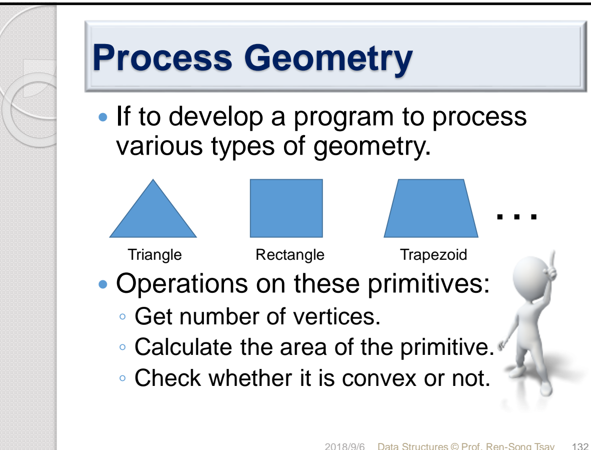
- An object A could be used to define another object B.
- Both objects and operations of A are automatically inherited by B.
  - `house` can inherit from `building`
- Advantages:
  - Simplification of software development
  - Easy to test and debug
  - Reusability
  - Flexibility

2018/9/6 Data Structures © Prof. Ren-Song Tsay 130

C++ Inheritance

- Define a class in terms of another class
  - easier to create and maintain an application.
- The existing class = **base** class, the new class = **derived** class.
- The **derived** class *is a* **base** class
  - dog IS-A mammal
- A class can be derived from more than one classes, i.e. inherit data and functions from multiple base classes.

2018/9/6 Data Structures © Prof. Ren-Song Tsay 131



Implement Using Classes

```
class Triangle {
public:
  Triangle ( ){
    m_VN = 3;
    mp_V = new Point [m_VN];
  }
  ~Triangle ( ){
    delete [] mp_V;
    mp_V = NULL;
  }
  double CalArea ( );
  bool isConvex();
  int vtxNum() { return m_VN; }
private:
  int m_VN;
  Point* mp_V;
};
```

```
class Rectangle {
public:
  Rectangle ( ){
    m_VN = 4;
    mp_V = new Point [m_VN];
  }
  ~Rectangle( ){
    delete [] mp_V;
    mp_V = NULL;
  }
  double CalArea ( );
  bool isConvex();
  int vtxNum() { return m_VN; }
private:
  int m_VN;
  Point* mp_V;
};
```


What's the Problem?

- Duplicated codes in both classes!
- If need to design 100 or more types of geometry?
- If need to add another "common" data member or function?
- Will need to maintain all the classes!

Polygon Base Class

- Polygon is an abstract type of geometry.
- Triangle and rectangle are special polygons.
- But how do we let the triangle and rectangle classes share the same code through using polygon?
- Don't worry! C++ will be your lifesaver!

2018/9/6 Data Structures © Prof. Ren-Song Tsay 135

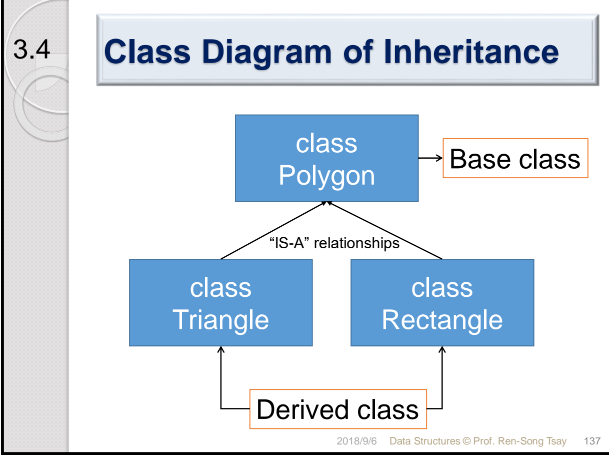
Class Inheritance in C++

- A mechanism to relate one class object to another one.
- Define a "IS-A" relationships between objects.
  - ***Type B* IS-A** data type of ***Type A*** if B is a **specialized** version of A and A is more **general** than B, e.g., Triangle IS-A Polygon and Rectangle IS-A Polygon.
- Members (data and functions) in Type A are implicitly copied to Type B.
- Reusability of code

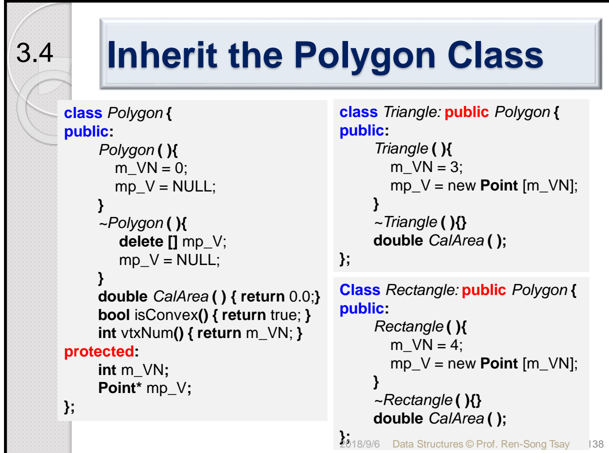
2018/9/6 Data Structures © Prof. Ren-Song Tsay 136

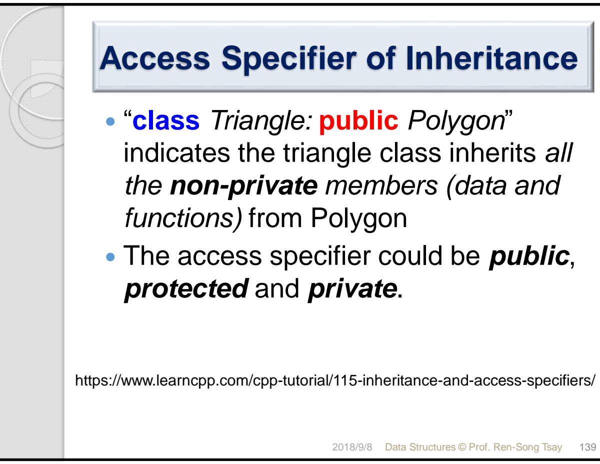
3.4

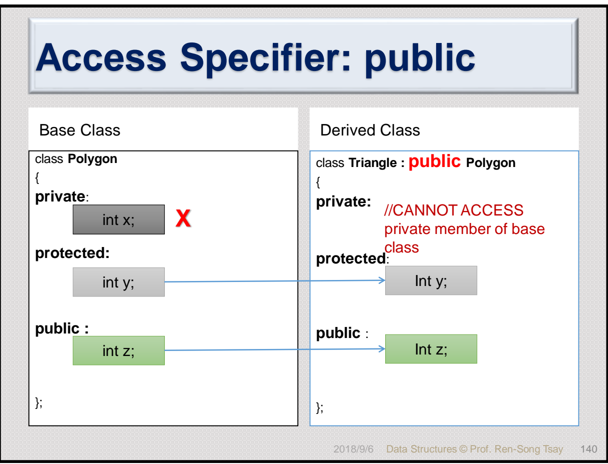
Class Diagram of Inheritance

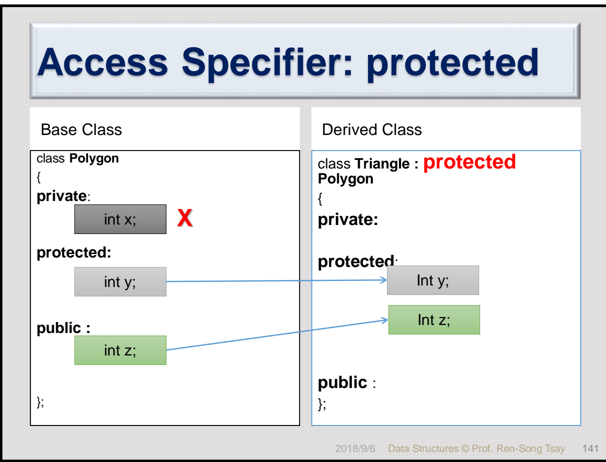


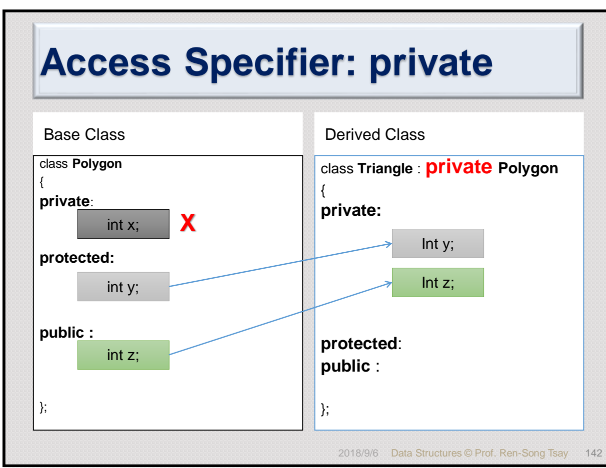
2018/9/6 Data Structures © Prof. Ren-Song Tsay 137

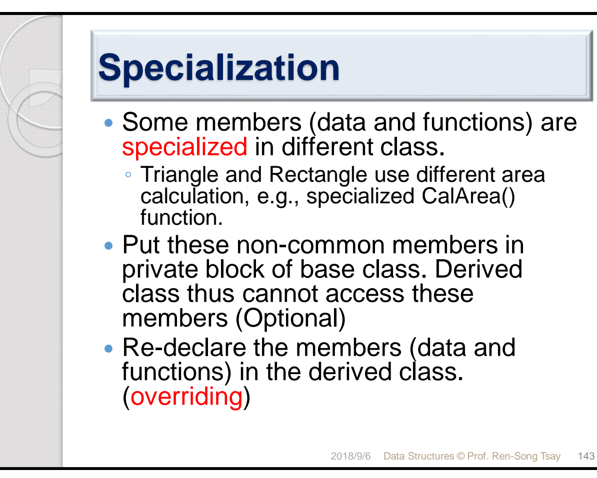

3.4
Inherit the Polygon Class
class Polygon {
public:
Polygon ( ){
m_VN = 0;
mp_V = NULL;
}
~Polygon ( ){
delete [] mp_V;
mp_V = NULL;
}
double CalArea ( ) { return 0.0;}
bool isConvex() { return true; }
int vtxNum() { return m_VN; }
protected:
int m_VN;
Point* mp_V;
};
class Triangle: public Polygon {
public:
Triangle ( ){
m_VN = 3;
mp_V = new Point [m_VN];
}
~Triangle ( ){}
double CalArea ( );
};
Class Rectangle: public Polygon {
public:
Rectangle ( ){
m_VN = 4;
mp_V = new Point [m_VN];
}
~Rectangle ( ){}
double CalArea ( );
};
2018/9/6 Data Structures © Prof. Ren-Song Tsay 138

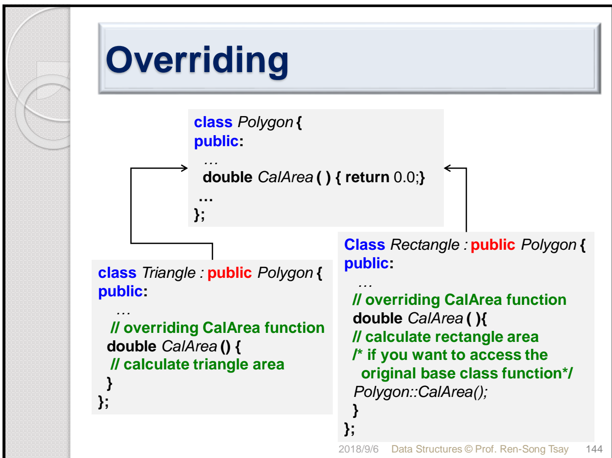

Access Specifier of Inheritance
"class Triangle: public Polygon" indicates the triangle class inherits all the non-private members (data and functions) from Polygon
The access specifier could be public, protected and private.
https://www.learncpp.com/cpp-tutorial/115-inheritance-and-access-specifiers/
2018/9/8 Data Structures © Prof. Ren-Song Tsay 139

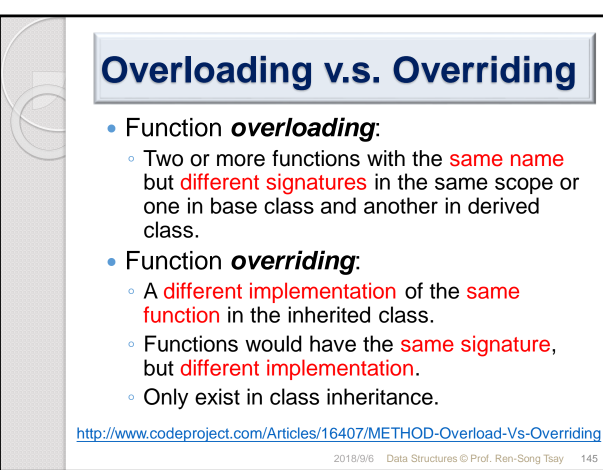

Access Specifier: public
Base Class
class Polygon
{
private:
int x;
X
protected:
int y;
public :
int z;
};
Derived Class
class Triangle : public Polygon
{
private:
//CANNOT ACCESS private member of base class
protected:
Int y;
public :
Int z;
};
2018/9/6 Data Structures © Prof. Ren-Song Tsay 140

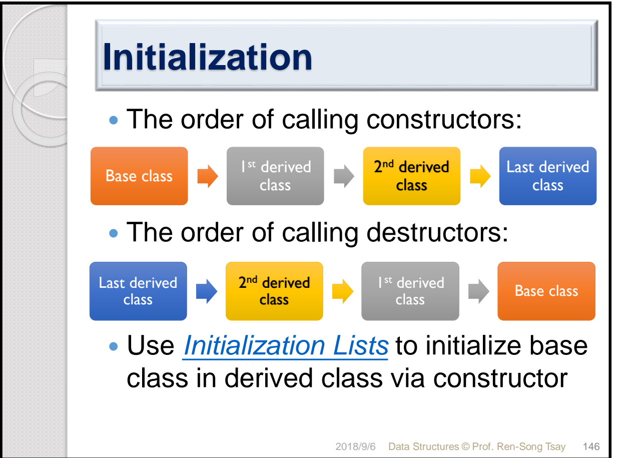

Access Specifier: protected
Base Class
class Polygon
{
private:
int x;
X
protected:
int y;
public :
int z;
};
Derived Class
class Triangle : protected Polygon
{
private:
protected:
Int y;
Int z;
public :
};
2018/9/6 Data Structures © Prof. Ren-Song Tsay 141

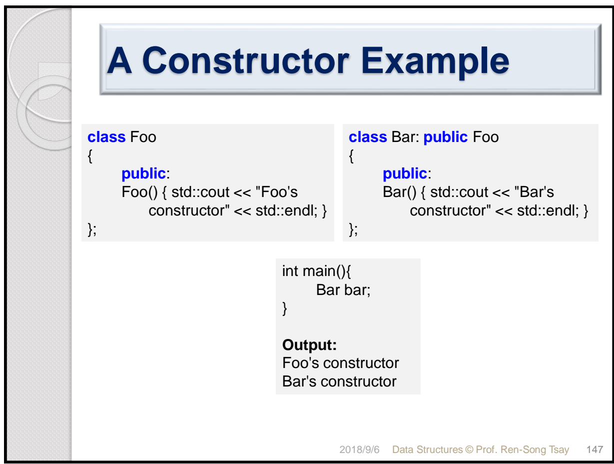

Access Specifier: private
Base Class
class Polygon
{
private:
int x;
X
protected:
int y;
public :
int z;
};
Derived Class
class Triangle : private Polygon
{
private:
Int y;
Int z;
protected:
public :
};
2018/9/6 Data Structures © Prof. Ren-Song Tsay 142

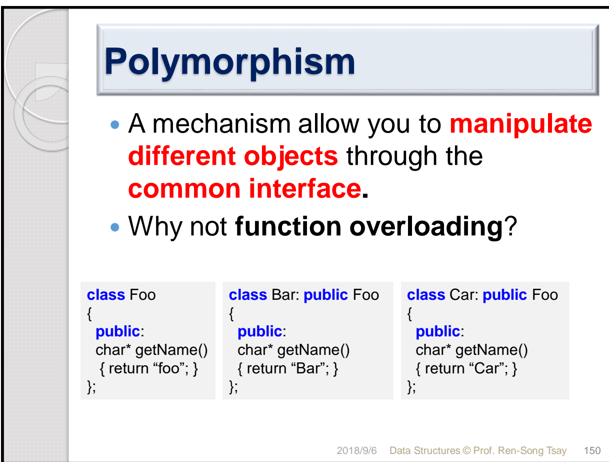

Specialization
Some members (data and functions) are specialized in different class.
Triangle and Rectangle use different area calculation, e.g., specialized CalArea() function.
Put these non-common members in private block of base class. Derived class thus cannot access these members (Optional)
Re-declare the members (data and functions) in the derived class. (overriding)
2018/9/6 Data Structures © Prof. Ren-Song Tsay 143

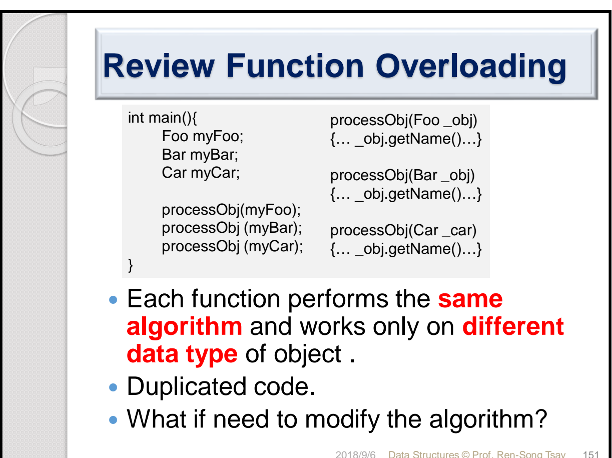

Overriding
class Polygon {
public:
…
double CalArea ( ) { return 0.0;}
…
};
class Triangle : public Polygon {
public:
…
// overriding CalArea function
double CalArea () {
// calculate triangle area
}
};
Class Rectangle : public Polygon {
public:
…
// overriding CalArea function
double CalArea ( ){
// calculate rectangle area
/* if you want to access the
original base class function*/
Polygon::CalArea();
}
};
2018/9/6 Data Structures © Prof. Ren-Song Tsay 144

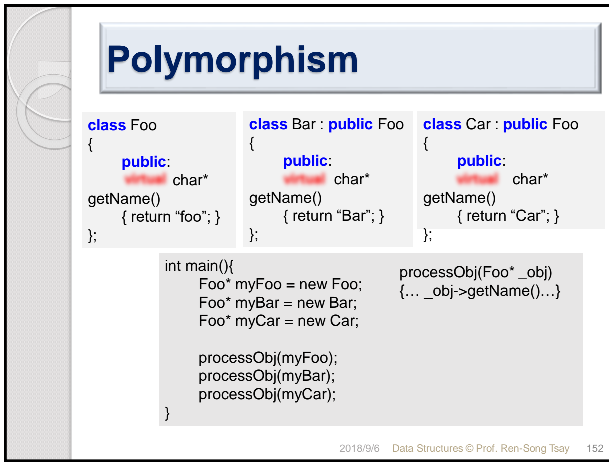

Overloading v.s. Overriding
Function overloading:
Two or more functions with the same name but different signatures in the same scope or one in base class and another in derived class.
Function overriding:
A different implementation of the same function in the inherited class.
Functions would have the same signature, but different implementation.
Only exist in class inheritance.
http://www.codeproject.com/Articles/16407/METHOD-Overload-Vs-Overriding
2018/9/6 Data Structures © Prof. Ren-Song Tsay 145


Initialization
The order of calling constructors:
Base class
1st derived class
2nd derived class
Last derived class
The order of calling destructors:
Last derived class
2nd derived class
1st derived class
Base class
Use Initialization Lists to initialize base class in derived class via constructor
2018/9/6 Data Structures © Prof. Ren-Song Tsay 146


A Constructor Example
class Foo
{
public:
Foo() { std::cout << "Foo's constructor" << std::endl; }
};
class Bar: public Foo
{
public:
Bar() { std::cout << "Bar's constructor" << std::endl; }
};
int main(){
Bar bar;
}
Output:
Foo's constructor
Bar's constructor
2018/9/6 Data Structures © Prof. Ren-Song Tsay 147


Polymorphism
A mechanism allow you to manipulate different objects through the common interface.
Why not function overloading?
class Foo
{
public:
char* getName()
{ return "foo"; }
};
class Bar: public Foo
{
public:
char* getName()
{ return "Bar"; }
};
class Car: public Foo
{
public:
char* getName()
{ return "Car"; }
};
2018/9/6 Data Structures © Prof. Ren-Song Tsay 150


Review Function Overloading
int main(){
Foo myFoo;
Bar myBar;
Car myCar;
processObj(myFoo);
processObj (myBar);
processObj (myCar);
}
processObj(Foo _obj)
{… _obj.getName()…}
processObj(Bar _obj)
{… _obj.getName()…}
processObj(Car _car)
{… _obj.getName()…}
Each function performs the same algorithm and works only on different data type of object .
Duplicated code.
What if need to modify the algorithm?
2018/9/6 Data Structures © Prof. Ren-Song Tsay 151



Polymorphism

```
class Foo
{
    public:
    virtual char*
getName()
    { return "foo"; }
};

class Bar: public Foo
{
    public:
    virtual char*
getName()
    { return "Bar"; }
};

class Car: public Foo
{
    public:
    virtual char*
getName()
    { return "Car"; }
};

int main(){
    Foo* myFoo = new Foo;
    Foo* myBar = new Bar;
    Foo* myCar = new Car;

    processObj(myFoo);
    processObj(myBar);
    processObj(myCar);
}

processObj(Foo* _obj)
{… _obj->getName()…}
```

2018/9/6 Data Structures © Prof. Ren-Song Tsay 153