Hochiminh City University of Technology
Computer Science and Engineering
[CO1027] - Fundamentals of C++ Programming

Operator Overloading & Inheritance

Lecturer: Duc Dung Nguyen
Credits: 3

Outline

- Operator overloading
- Friendship
- Inheritance

Operator overloading

Fundamentals of Operator Overloading

- Overloading an operator
  - –Write function definition as normal
  - –Function name is keyword **`operator`** followed by the symbol for the operator being overloaded
  - –**`operator+`** used to overload the addition operator (**`+`**)
- Using operators
  - –To use an operator on a class object it must be overloaded unless the assignment operator **`(=)`** or the address operator **`(&)`**
    - Assignment operator by default performs memberwise assignment
    - Address operator (&) by default returns the address of an object

Restrictions on Operator Overloading

Operators that can be overloaded							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

Restrictions on Operator Overloading

- Overloading restrictions
  - –Precedence of an operator cannot be changed
  - –Associativity of an operator cannot be changed
  - –Arity (number of operands) cannot be changed
    - Unary operators remain unary, and binary operators remain binary
    - Operators `&`, `*`, `+` and `-` each have unary and binary versions
    - Unary and binary versions can be overloaded separately
- No new operators can be created
  - –Use only existing operators
- No overloading operators for built-in types
  - –Cannot change how two integers are added
  - –Produces a syntax error

Friendship

Friendship

- Friends are functions or classes declared with the friend keyword.
- Using friend functions can enhance performance.

Friend function member

- A non-member function can access private and protected members of class if it is declared as a friend of class.
- E.g.:

```
class Student {
```

- `. . .`

```
public:
    friend Student duplicateStudent(Student& a);
};
```

Friend class

- Friend class: is a class whose members can access to private and protected members of other classes.

- ```
  class Lecturer;
  class Student {
       friend class Lecturer;// lecturer is a friend
       . . .
  };
  ```
  ```

Inheritance

What Is Inheritance?

- Provides a way to create a new class from an existing class
- The new class is a specialized version of the existing class

Advantages of inheritance

- When a class inherits from another class, there are three benefits:
  - You can reuse the methods and data of the existing class
  - You can extend the existing class by adding new data and new methods
  - You can modify the existing class by overloading its methods with your own implementations

The ″is a″ Relationship

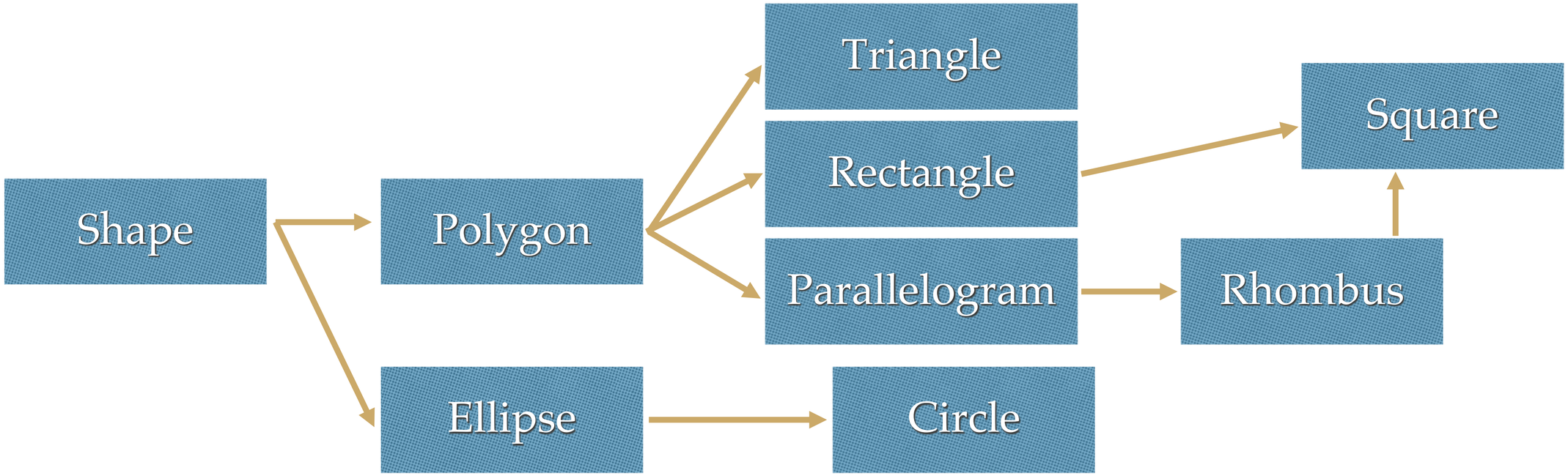
- Inheritance establishes an "is a" relationship between classes.
  - A poodle is a dog
  - A car is a vehicle
  - A flower is a plant
  - A football player is an athlete

Inheritance – Terminology and Notation in C++

- Base class (or parent) – inherited from
- Derived class (or child) – inherits from the base class
- Notation:

```
class Student          // base class
{
 . . .
};
class UnderGrad : public student
{                 // derived class
 . . .
};
```

Inheritance Exmaple



Inheritance Syntax

- class <CName> [: <access specifier> <BaseCName>] {
  . . .
  };
- E.g.:

```
class Polygon : public Shape { . . . };
class Rectangle : public Polygon { . . . };
class Square : public Rectangle, public Rhombus { . . . };
class Ellipse: public Shape { . . . };
```

Inheritance Example

```
class Shape {
    int    id;
public:
    Shape() { id = 0; }
    ~Shape();
    void draw();
};
```

```
#include "Shape.h“
class Polygon : public Shape {
    int        nVertex;
    Vector2D* pVertex;
public:
    Polygon(int n) : Shape(), nVertex(n) {}
    ~Polygon();
    void draw();
};
```

```
#include "Shape.h“
class Ellipse : public Shape {
    float     theta;
    Vector2D  center, len;
public:
    Ellipse();
    ~Ellipse();
    void draw();
};
```

Back to the ‘is a’ Relationship

- An object of a derived class 'is a(n)' object of the base class
- Example:
  - an `UnderGrad` is a `Student`
  - a `Mammal` is an `Animal`
- A derived object has **all** of the characteristics of the base class

What Does a Child Have?

An object of the derived class has:

- all members defined in child class
- all members declared in parent class

An object of the derived class can use:

- all `public` members defined in child class
- all `public` members defined in parent class

Rules for building a class hierarchy

- Derived classes are special cases of base classes
- A derived class can also serve as a base class for new classes.
- There is no limit on the depth of inheritance allowed in C++ (as far as it is within the limits of your compiler)
- It is possible for a class to be a base class for more than one derived class

Constructors and Destructors in Base and Derived Classes

- Derived classes can have their own constructors and destructors
- When an object of a derived class is created, the base class's constructor is executed first, followed by the derived class's constructor
- When an object of a derived class is destroyed, its destructor is called first, then that of the base class

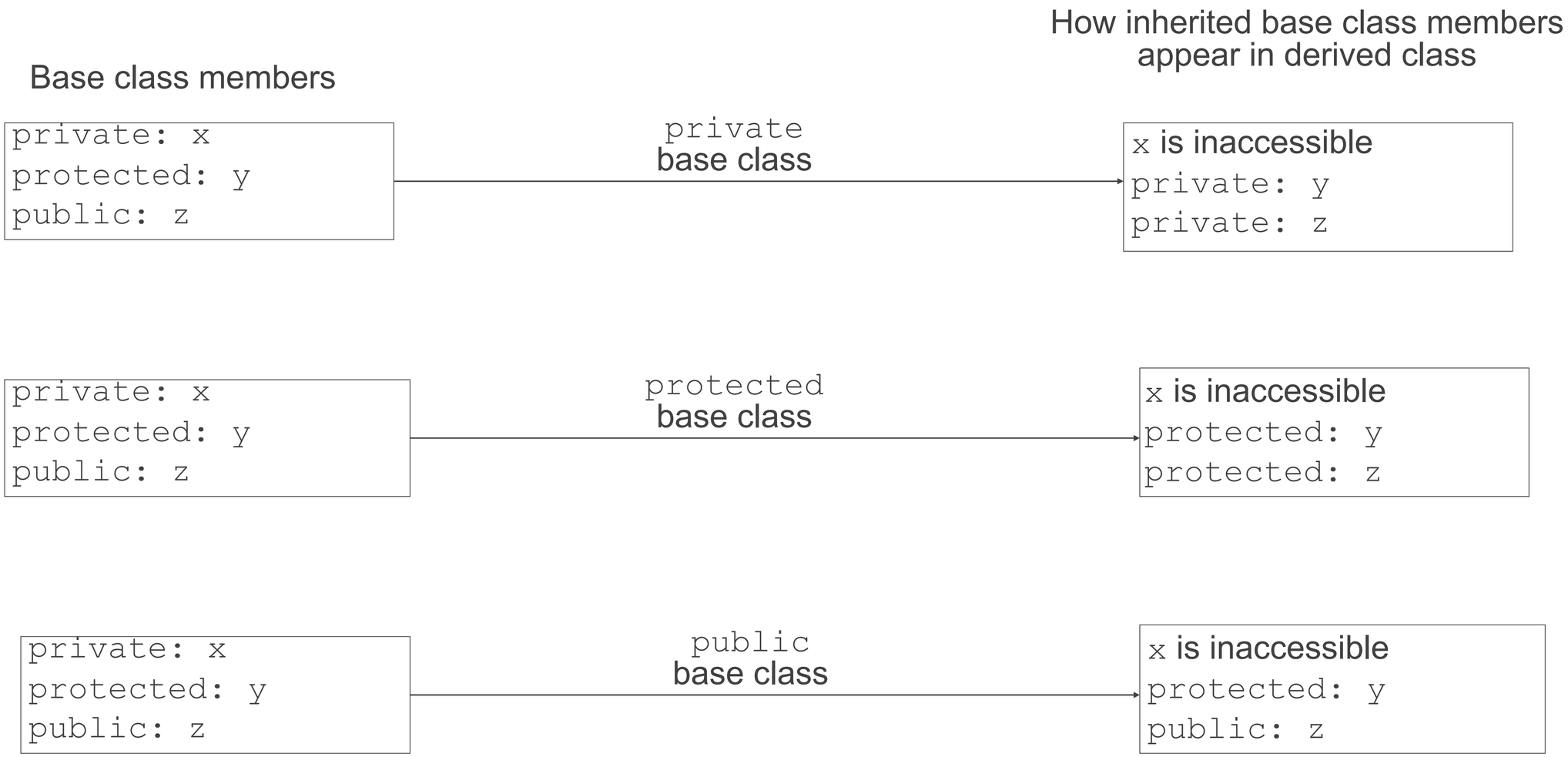
Passing Arguments to Base Class Constructor

- Allows selection between multiple base class constructors
- Specify arguments to base constructor on derived constructor heading:
  - E.g: Square::Square(int side) : Rectangle(side, side)
- Can also be done with inline constructors
- Must be done if base class has no default constructor

Protected Access

- A base class's **public** members are accessible within its body and anywhere that the program has a handle to an object of that class or one of its derived classes.
- A base class's **private** members are accessible only within its body and to the friends of that base class.
- A base class's **protected** members can be accessed within the body of that base class, by members and friends of that base class, and by members and friends of any classes derived from that base class.
- Using **protected** access offers an **intermediate level** of protection between **public** and **private** access.

Inheritance vs. Access



Inheritance vs. Access

Base-class member-access specifier	Type of inheritance		
	public inheritance	**protected** inheritance	**private** inheritance
public	**public** in derived class. Can be accessed directly by member functions, **friend** functions and nonmember functions.	**protected** in derived class. Can be accessed directly by member functions and **friend** functions.	**private** in derived class. Can be accessed directly by member functions and **friend** functions.
protected	**protected** in derived class. Can be accessed directly by member functions and **friend** functions.	**protected** in derived class. Can be accessed directly by member functions and **friend** functions.	**private** in derived class. Can be accessed directly by member functions and **friend** functions.
private	Hidden in derived class. Can be accessed by member functions and **friend** functions through **public** or **protected** member functions of the base class.	Hidden in derived class. Can be accessed by member functions and **friend** functions through **public** or **protected** member functions of the base class.	Hidden in derived class. Can be accessed by member functions and **friend** functions through **public** or **protected** member functions of the base class.

Summarise

- Operator overloading
- Friendship
- Inheritance