IM 1003: Computer Programming	
Classes (Part 2)	
Ling-Chieh Kung	
Department of Information Management	
National Taiwan University	
Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 – Classes (Part 2)	1 / 27

Static members

- A member variable/function may be an attribute/operation of a class, not an object.
- This happens when the attribute/operation is **class-specific** rather than object-specific.
 - It should be identical for all objects of this class.
- These variables/functions are called **static members**.

Outline

- Static members
- Objects and pointers
- friend
- Destructors

 Ling-Chieh Kung
 NTU IM

 Programming Design , Spring 2013 – Classes (Part 2)
 2 / 27

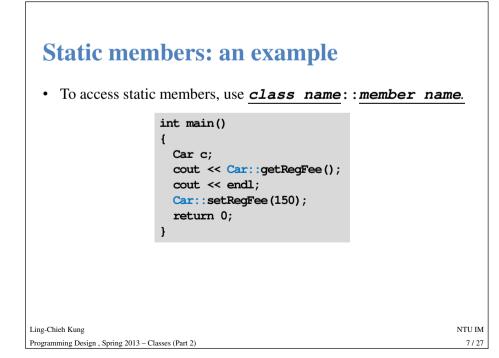
Static members: an example

- The annual registration fee for all cars (at least for those in the same type) is identical.
- The member functions that access only static member variables can be set static.

class Car

```
{
private:
    // ...
static int regFee;
    // static variable declaration
public:
    // ...
static int getRegFee();
static void setRegFee();
    // static function declaration
};
```

Static members: an example		Static members: an example
 You cannot initialize a static variable inside the class definition. You have to initialize a static variable. You have to initialize a static variable globally. 	<pre>double Car::regFee = 200; // static variable initialization double Car::getRegFee() { return Car::regFee; // or return regFee; } void Car::setRegFee(int regFee) { Car::regFee = regFee; }</pre>	<pre>• For static functions, you may still write class Car { // public: // static double getRegFee() { return Car::regFee; } static void setRegFee(int regFee) { Car::regFee = regFee; } }</pre>
		 You just cannot assign values to static (actually any) variables class definition block.
Ling-Chieh Kung	NTU IM	Ling-Chieh Kung
Programming Design, Spring 2013 – Classes (Part 2)	5/27	Programming Design, Spring 2013 – Classes (Part 2)



Static members: an example

Static members: an example

	class Car { //	
	public:	
	<pre>// static double getRegFee() { return Car::regFee; } static void setRegFee(int regFee) { Car::regFee = regFee; } }</pre>	
	ust cannot assign values to static (actually any) variables inside definition block.	the
Chieh Kung		NTU IM
amming Design , Sp	pring 2013 – Classes (Part 2)	6/27

Static members

- Recall that we have four types of members:
 - Instance variables and instance functions.
 - Static variables and static functions.
- Some rules regarding static members:
 - You cannot access an instance member inside a static function. Why?
 - You may access a static member inside an instance function.
 - Though not suggested, you may access a static member through an object.

Car c; cout << c.getRegFee() << endl;</pre>

Good programming

- If one attribute should be identical for all objects, it should be declared as a static variable.
 - Do not make it an instance variable and try to maintain consistency.
- Do not use an object to invoke a static member.
 - This will confuse the reader.
- Use *class name*: : *member name* even inside member function definition to show that it is a static member.
 - The reason is the same as using **this**->.

Ling-Chieh Kung	NTU IM
Programming Design, Spring 2013 - Classes (Part 2)	9/27

Objects and pointers

- You can use a pointer to point to an object.
- This can be more useful than pointing to a basic data type.
- One of those important reasons is that passing a pointer is **more efficient** than passing the whole object.
 - A pointer is **smaller** than an object.
 - Copying a pointer is easier than **copying an object**.
 - Copying an object requires one to be careful!
- Other reasons will be discussed in other lectures.

Outline

- Static members
- Objects and pointers
- friend
- Destructors

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 2)	10 / 27

Objects and pointers

- When you use a pointer to access an object, you may use "->".
- Otherwise, you have to use "*", such as (*ptrA).print().

Point* prtA = &A; // a pointer to an object
(*ptrA).print(); // A(10, 20)
ptrA->print(); // A(10, 20)

Passing objects into a function

- Consider a function that
 - Takes three points as the input and returns the center of gravity.

```
Point cenGrav(Point p1, Point p2, Point p3, char n)
{
    double x = (p1.getX() + p2.getX() + p3.getX()) / 3;
    double y = (p1.getY() + p2.getY() + p3.getY()) / 3;
    Point cog(x, y, n);
    return cog;
}
```

- We need to create **four Point** objects in this function.
- Example "12_01_objectPointer": The constructor is invoked four times!

Ling-Chieh Kung	NTU IM
Programming Design, Spring 2013 – Classes (Part 2)	13/27

Copying an object

- When do we copy an object?
 - When we pass an object into a function using the call-by-value mechanism.
 - When we assign an object to another object.
 - When we create an object with another object as the argument of the constructor.
- When an object is created by copying another object, the **copy constructor** will be invoked.
 - If the programmer does not define one, the compiler add a default copy constructor into the class.
 - The default copy constructor simply copies all member variables one by one, no matter a variable is of a basic data type, an array, a pointer, or an object.
 - Example "12_02_copyConstructor".

Passing object pointers into a function

• We may rewrite this function and **pass pointers** rather than objects into this function:

```
Point cenGrav(Point* p1, Point* p2, Point* p3, char n)
{
    double x = (p1->getX() + p2->getX() + p3->getX()) / 3;
    double y = (p1->getY() + p2->getY() + p3->getY()) / 3;
    Point cog(x, y, n);
    return cog;
}
    - We need to create only one Point object in this function.
    Example "12_01_objectPointer".
    Nevertheless, using pointers to access members requires more time.
Ling-Chich Kung NTU M
Programming Design, Spring 2013-Clases (Part 2) 14/27
```

Copy constructors

- We may implement our own copy constructor.
- The input of a copy constructor must be a **constant reference**.

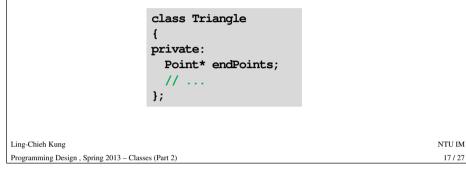
```
Point::Point(const Point& p)
{
    x = p.x;
    y = p.y;
    name = p.name;
}
```

- This has nothing different from the default copy constructor.

Ling-Chieh Kung	
Programming Design , Spring 2013 - Classes (Part 2)	

Shallow copy

- If no member variable is an array or a pointer, a default copy constructor is enough.
- If there is any array or pointer member variable, the default copy constructor does a "**shallow copy**".
- Consider a class **Triangle** containing a **Point** array.



Deep copy

- To correctly copy a triangle (by creating new points), we need to write our own copy constructor.
- We say that we need to implement a "deep copy" by ourselves.
 - In the self-defined copy constructor, we manually create points, set their values with the original points, and use **endpoint** to point to them.
 - Example "12_04_deepCopy".

Shallow copy

- Suppose in **Triangle t2**, **t2**.**endPoint** currently points to three points (through, e.g., **endPoint = new Point[3]**).
- Suppose we adopt the default copy constructor.
- When we do **Triangle t3 = t2**:
 - t3.endPoint will point to the same three points!
 - The default copy constructor does not create new points for us. It simply copies the value of t2.endPoint to t3.endPoint.
 - Once a point is moved in one triangle, that point in the other triangle will also be moved!
 - Example "12_03_shallowCopy".

.ing-Chieh Kung	NTU IM
Programming Design, Spring 2013 – Classes (Part 2)	18 / 27

Shallow copy vs. deep copy

- A comparison between shallow copy and deep copy.
- Why not endPoint = t.endPoint; in deep copy?

```
Triangle::Triangle(const Triangle& t)
{
    endPoint = t.endPoint;
    name = t.name;
```

```
Triangle::Triangle(const Triangle& t)
{
  endPoint = new Point[3];
  for(int i = 0; i < 3; i++)
    endPoint[i] = t.endPoint[i];
  // endPoint = t.endPoint;
  name = t.name;
}</pre>
```

Ling-Chieh Kung

Outline

- Static members
- Objects and pointers
- friend
- Destructors

NTU IM
21 / 27

friend: an example void test() ł Point p; p.x = 100; // syntax error if not a friend cout << p.x; // syntax error if not a friend</pre> class Test public: void test (Point p) £ p.x = 200; // syntax error if not a friend cout << p.x; // syntax error if not a friend } }; Ling-Chieh Kung Programming Design, Spring 2013 - Classes (Part 2)

NTU IM

23/27

 One class can allow its "friends" to a Its friends can be global functions of Then inside test () and member functions of Test, those private members of Point can be accessed. Point cannot access Test's members. 	1
 A friend can be declared in the public A class must declare its friends by its 	•

Programming Design , Spring 2013 - Classes (Part 2)

22/27

<section-header><list-item><list-item><list-item><list-item><list-item><list-item><list-item><table-container>

Outline

- Static members
- Objects and pointers
- friend
- Destructors

NTU IM
25/27
_

Destructors

- One important mission to be done by a destructor is to release those dynamically-allocated memory spaces pointed by member variables.
 - The default destructor does not do this.
 - One must do this by herself/himself.
 - If this is not done, there will be memory leaks.

Tı	riangle:	:~!	<pre>Friangle()</pre>
{			
	delete	[]	endPoint;
}			

Destructors

- A destructor is invoked right before an object is destroyed. - It should be public and have no parameter.
- To replace the default destructor by a self-defined one, use ~:

<pre>class Point { // public: // // destructor ~Point() { cout << "Bye!\n"; } };</pre>	
Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 2)	26/27