IM 1003: Computer Programming	
Classes (Part 1)	
Ling-Chieh Kung	
Department of Information Management	
National Taiwan University	
Ling-Chieh Kung	NTU IM
Programming Design, Spring 2013 - Classes (Part 1)	1/42

Object-oriented programming

- Until now, we have focused on procedural programming.
- The keys in it are logical controls and subprocedures. In other words, **if**, **for**, and functions.
- We will begin to introduce a new programming methodology: **object-oriented programming (OOP)**.
- It is based on procedural programming.
- It is different from procedural programming from the perspective of thinking.

Outline

- Basic ideas
- Visibility and encapsulation
- this and that
- Constructors

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 1)	2/42

Classes and objects

- In C, we use structures; in C++, we use **classes**.
- Like structures, we can use classes to define data types by ourselves and create variables called **objects**.
- As we will see, classes are much more powerful than structures.

Classes and objects

- In a class, we can define variables and functions, just as we did in a structure.
 - They are call **member variables** and **member functions**.
- However, now there are four types of class members:
 - Instance variables (default).
 - Static variables.
 - Instance functions (default).
 - Static functions.

Ling-Chieh Kung

Programming Design, Spring 2013 - Classes (Part 1)

• Starting from now, when we say member variables (fields) and member functions, we are talking about instance ones.

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 1)	5/42

An example in struct

struct Point	int main()	
{	{	
int x;	Point A;	
int y;	A.name = 'A';	
char name;	A.x = 20;	
};	A.y = 30;	
<pre>void print (Point p);</pre>	print (A); // A(20, 30)	
void print (Point p)	return 0;	
{	}	
cout << p.name << "("		
<< p.x << ", "		
<< p.y << ")";		
}		
Ling-Chieh Kung		NTU IM
Programming Design , Spring 2013 - Classes (Part 1)		6/42

An example in class int main() { Point A; // an object A.name = 'A'; A.x = 20; A.y = 30; A.print(); // invoking instance function return 0; }

Ling-Chieh Kung Programming Design, Spring 2013 – Classes (Part 1)

NTU IM

7/42

Instance functions

- When using a class, we define instance functions **in the class** and invoke them through objects.
- Instance functions are functions that do something with this object's instance variables or functions.

- e.g., print().

NTU IM
9/42

Instance function definition

• We may also define the function inside the class definition.

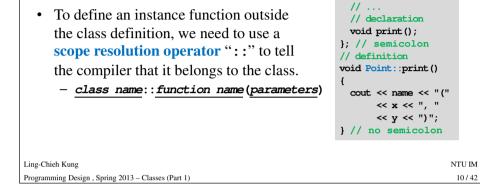
```
class Point
{
    // ...
    // declaration and definition
    void print()
    {
        cout << name << "("
            << x << ", "
            << y << ")";
        } // no semicolon
    }; // semicolon
Ling-Chich Kung
Programming Design, Spring 2013-Classes (Part 1)</pre>
```

NTU IM

11/42

Class definition

- Keyword: class.
- **Declare** instance function **in** the class definition, and then **define** the function **after** the class definition.
- A semicolon is needed.



Invoking instance functions

• In the main function, we use **A.print()** instead of **print(A)**.

```
int main()
{
    Point A; // an object
    // instance function invocation
    A.print();
    return 0;
}
```

• To invoke an instance function through an object, use ".".

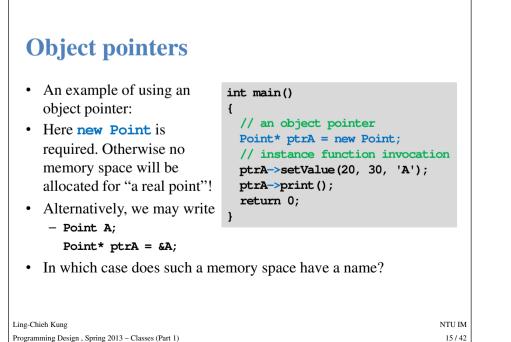
```
- object name.function name(arguments);
```

Ling-Chieh Kung
Programming Design, Spring 2013 - Classes (Part 1)

Instance functions with parameters

• Instance functions may also have parameters:

<pre>class Point { // void setValue(int, int, char); }; void Point::setValue(int a, int b, char c) { x = a; y = b; name = c; }</pre>	<pre>int main() { Point A; A.setValue(20, 30, 'A'); A.print(); return 0; }</pre>
Ling-Chieh Kung	NTU IM
Programming Design, Spring 2013 - Classes (Part 1)	13 / 42



Object pointers

- What we have done is to use an object to invoke instance functions.
- If we have a pointer **ptrA** pointing to the object **A**, you may use (*ptrA).print() to invoke the instance function print().
 - ***ptrA** returns the object **A**
- To simplify this, C++ creates the operator ->.
 - This is specifically for an object pointer to access its members.
 - (*ptrA).print() is equivalent to ptrA->print().
 - (*ptrA) .x is equivalent to ptrA->x.

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 1)	14 / 42

Invoking instance functions in classes

• In an instance function, you can invoke another instance function (or itself recursively).

<pre>void Point::printDistance() { cout << distance(); } double Point::distance() { double a = static_cast<double>(x); } </double></pre>	<pre>class Point { // double distance(); void printDistance(); };</pre>
<pre>double b = static_cast<double>(y); return sqrt(a * a + b * b); // <cmath> }</cmath></double></pre>	

Ling-Chieh Kung Programming Design , Spring 2013 - Classes (Part 1)

NTU IM

16/42

Outline

- Basic ideas
- Visibility and encapsulation
- this and that
- Constructors

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 1)	17 / 42

Visibility

• A class with different visibility levels:

<pre>class Point { private: // private members int x; int y; char name; public: // public members void setValue(int a, int b, char n); void print(); };</pre>	<pre>int main() { Point A; A.setValue(20, 30, 'A'); // A.x = 20 is wrong A.print(); return 0; }</pre>	
 Private instance members can only be accessed inside the definition of instance functions. 		
- Now, is it allowed to have a point with only the <i>x</i> -coordinate?		
Ling-Chieh Kung	NTU IM	
Programming Design, Spring 2013 - Classes (Part 1)	19/42	

Visibility

- We can set visibility of members in a class:
 - public: it can be accessed anywhere.
 - **private**: it can be accessed only **in the class**.
 - **protected**: discussed later in this semester.
- These three keywords are the visibility modifiers.
- If we remove the **public** modifier, the program will not run.
 - It is because the default level of visibility is **private**.
- By setting visibility, we can hide our instance members (usually variables).
 - Before we ask why, let's see how to set visibility.

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 1)	18 / 42

Why data hiding?

- In general, when we write a class, we want it to work as we expect.
 That is, "under control".
- For example, we do not want a point to be printed out in strange formats, such as A{20, 30}, A:20, 30, etc.
- If we allow one to access **x**, **y**, and **name** in the main function, he can print out a point in any way he likes!
- To prevent this, we set instance variables to be private and leave **print ()** public. When one (typically another programmer) wants to print out a point, the only available format is A(20, 30).

Ling-Chieh Kung Programming Design , Spring 2013 – Classes (Part 1)

Why data hiding? Another example

	class BankAccount	
	public:	
	int balance;	
	void deposit (int amount)	
	void withdraw(int amount)	
	bool transferTo(int amount, BankAccount to);	
	};	
	void BankAccount::deposit(int amount)	
	{	
	balance += amount;	
	}	
	void BankAccount::withdraw(int amount)	
	{	
	balance -= amount;	
	}	
Ling-Chieh Kung		NTU IM
Programming Design ,	Spring 2013 – Classes (Part 1)	21/42

Why data hiding? Another example

- Suppose another programmer needs to use the class **BankAccount** to write an ATM program.
- If **balance** is public, we can not predict what will be done on balance.
 - The programmer may modify balance in a wrong way.
 - E.g., he may write a transfer function without checking the balance!
- As the developer of a class, we should set **balance** private and design/implement appropriate member functions for others to use.

Why data hiding? Another example

	<pre>bool BankAccount::transferTo(int amount, BankAccount to) { if (amount >= balance) { withdraw(amount); to.deposit(amount); return true; } else return false; }</pre>	
Ling-Chieh Kung	g	NTU IM
Programming De	esign, Spring 2013 – Classes (Part 1)	22/42

Visibility

- In general, some instance variables/functions should not be accessed directly (or even known) by other ones. They should be used only in the class.
- In this case, set them private. ٠
- You may see many classes with all instance variables private and all instance functions public.
 - If you do not know what to do, do this.
 - However, any instance function that should not be invoked by others should also be private.

Private instance functions

• In the following example, if **distance()** is not allowed to be invoked by others, it should be private.

<pre>void Point::printDistance() { cout << distance(); } double Point::distance() { double a = static_cast<double>(x); double b = static_cast<double>(y); return sqrt(a * a + b * b); }</double></double></pre>	<pre>class Point { private: // double distance(); public: void setValue(int x, int y, char name); void print(); void printDistance(); };</pre>	
Ling-Chieh Kung NTU IM		
Programming Design, Spring 2013 – Classes (Part 1)	25/42	

Outline

- Basic ideas
- Visibility and encapsulation
- this and that
- Constructors

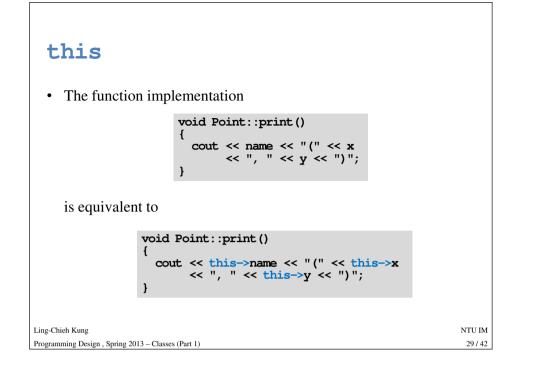
Encapsulation

- The concept of **packaging** (member variables and member functions) and **data hiding** is together called "**encapsulation**".
 - Roughly speaking, we pack data (member variables) into a black box and provide only controlled interfaces (member functions) for others to access these data.
 - Others should not even know how those interfaces are implemented.
- For OOP, there are three main characteristics/functionalities:
 - Encapsulation.
 - Inheritance.
 - Polymorphism.
- The last two will be discussed later in this semester.

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 1)	26 / 42

this

- When you create an object, it occupies a memory space and has an address.
- this is a pointer storing the address of the object.
 this is a C++ keyword.
- When the compiler reads **this**, it looks at the memory space to find the object you are referring to.



Good programming style

- You may choose to always use **this**-> when accessing instance variables and functions.
- This will allow other programmers (or yourself in the future) to know they are members without looking at the class definition.

Ling-Chieh Kung Programming Design , Spring 2013 – Classes (Part 1)

this

do {

dc {

}

- Suppose **x** is an instance variable.
 - Usually you can use **x** directly instead of **this->x**.
 - However, if you want to have a local variable or function parameter having the same name with an instance variable, you need this->.

```
void Point::setValue(int x, int y, char name)
{
    this->x = x;
    this-> y = y;
    this-> name = name;
    }
    A local variable hides the instance variable with the same name.
    - this->x: the instance variable, x: the local variable.
Ling-Chich Kung NTU M
Programming Design, Spring 2013 - Classes (Part 1) 30/42
```

Instance function overloading

• You can overload an instance function with different parameters as well as what we did for global functions.

ouble Point::distance()	class Point
	{
<pre>double a = static_cast<double>(x);</double></pre>	private:
<pre>double b = static_cast<double>(y);</double></pre>	//
return sqrt (a * a + b * b);	double distance();
	double distance(Point to);
ouble Point::distance(Point to)	public:
	//
<pre>double a = static_cast<double>(x - to.x);</double></pre>	<pre>void print();</pre>
<pre>double b = static_cast<double>(y - to.y);</double></pre>	<pre>void printDistance();</pre>
return sqrt(a * a + b * b);	<pre>void printDistance(Point to);</pre>
	};

Ling-Chieh Kung	
Programming Design , Spring 2013 - Classes (Part 1)	

Objects as parameters or return values

- You can pass an object into any function as well as what we did with structures.
- A function can return an object.
- Point vector (Point p1, Point p2);
 - This should be a global function rather than an instance function. Why?

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 1)	33 / 42

Object arrays

• You can create an array whose elements are objects.

class Triangle
{
private:
<pre>Point endPoints[3];</pre>
//
};

Objects as instance variables

- A instance variable's type can be a class.
- In other words, an object can have other objects as members.
 Recall that this can also happen for structures.
- As an example, we may define a class **Triangle** that contains three **Point** objects.

	<pre>class Triangle { private: Point point1; Point point2; Point point3; // };</pre>	
Ling-Chieh Kung		NTU IM
Programming Design , Spring 2013 – Classes (Part 1)		34/42

Outline

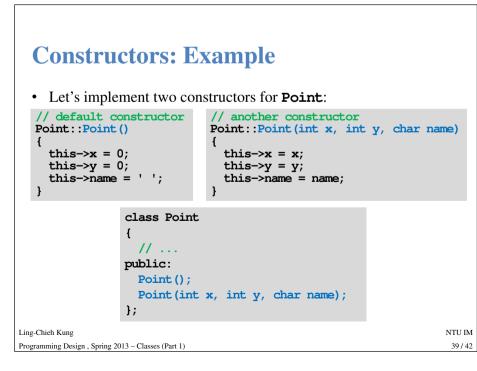
- Basic ideas
- Visibility and encapsulation
- this and that
- Constructors

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 – Classes (Part 1)	36 / 42

Constructors

- It is an **instance function** of a class.
 - However, it is very **special**.
- A constructor will be invoked **automatically** when the object is **created**.
 - It must be invoked.
 - It cannot be invoked twice.
 - It cannot be invoked by the programmer manually.
- Usually it is used to initialize the object.

Ling-Chieh Kung	NTU IM
Programming Design , Spring 2013 - Classes (Part 1)	37 / 42



Constructors

- A constructor's named is the same as the class.
- It does not return anything, even **void**.
- You can (and usually you will) overload constructors.
- The constructor with **no parameter** is the **default constructor**.
- If a programmer does not define any constructor, the **compiler** makes a default one which **does nothing**.
 - Once the programmer defines a constructor (with or without parameters), the complier will **not** create a default constructor.
- A constructor may be private.
 - In this course, you probably will not need to have private constructors.

Ling-Chieh Kung	NTU IM
Programming Design, Spring 2013 - Classes (Part 1)	38 / 42

Constructors: Example

• Now, when we create objects:

```
int main()
{
    Point A(10, 15, 'A');
    A.print(); // A(10, 15)
    A.printDistance(); // 18.0278
    Point B;
    B.print(); // (0, 0)
    B.printDistance(); // 0
    return 0;
}
```

- Example "11_01_Point".

Ling-Chieh Kung	NTU IM
Programming Design, Spring 2013 - Classes (Part 1)	40 / 42

Good programming style

- If any member variable needs an initial value when an object is created, you should write a constructor to initialize it.
- Use constructor overloading to provide flexibility of initializing member variables.

Timing for invoking constructors

- When a class has other classes as types of instance variables, when do all the constructors be invoked?
 - Example "11_02_Triangle".

Ling-Chieh Kung	NTU IM
Programming Design, Spring 2013 – Classes (Part 1)	41/42

 Ling-Chieh Kung
 NTU IM

 Programming Design , Spring 2013 – Classes (Part 1)
 42 / 42