

# Lecture 3

## Classes and Objects

# Outline

- Classes and Objects
- Dynamically allocated Objects
- Arrays of Objects
- Controlling Access to Member Attributes
- Friend Classes
- Friend Functions
- The this Pointer
- Function returning an object

# Classes and Objects

- Real world object : Attributes & Abilities
- Programming object : Data & Functions
- **Class** is a data type which is used to define objects.
- A class serves as a model description.
- It specifies what **data** and what **functions** will be included in **objects** of that class.

# Classes and Objects

- A **class** is a grouping of data and functions.
- A class is very much like a **struct** type as used in C.
- Writing a class doesn't create any objects.
- An **object** is an instance of a class, which is similar to a **variable**.
- An object is what you actually use in a program.
- An **attribute** is a **member data** of a class.
- Example; Name of a student, coordinates of a point.
- A **method (message)** is a **member function** contained within class.

# Example : Point class

- A Point class can be written in a graphics program.
- The Point class will have two properties:
  - **x and y coordinates.**
- The Point class can have the following functions :
  - **move function** : Moves to a location.
  - **print function** : Shows the coordinates on screen.
  - **is\_zero function** : Checks whether it is on the zero point (0, 0) or not.


# Point class declaration

- The Point class contains x and y member data. Their access type is private by default.
- Class also contains member functions, which are declared as public access.

	<b>Class</b> Point	<i>// Declaration of Point Class</i>
	{	
Datas	int x, y;	<i>// Current x and y coordinates</i>
	public:	<i>// public access allowed</i>
Functions (prototypes)	void move(int, int);	<i>// A function to move to a new point</i>
	void print();	<i>// To print the coordinates on screen</i>
	bool is_zero();	<i>// Is the point on the zero point (0,0) ?</i>
	};	

# Member Functions of Point class

`::` is scope operator. The move function is a member of Point class.



```
// A function to move the points
void Point :: move (int new_x, int new_y)
{
    x = new_x;    // assigns new value to x coordinate
    y = new_y;    // assigns new value to y coordinate
}
```

```
// Print the coordinates on the screen
void Point :: print ()
{
    cout << "X= " << x << ", Y= " << y << endl;
}
```

```
// Check whether the point is on the zero point(0,0)
bool Point :: is_zero ()
{
    return (x == 0) && (y == 0);
    // returns true or false
}
```

# Defining Point Objects

In main program, we can define variables (objects) by using the Point class.

```
int main()
{
    Point P1, P2; // Two object variables are defined.

    P1.move(100, 50);      // P1 moves to (100, 50) location
    P1.print();
    if ( P1.is_zero() )
        cout << "P1 is on zero point" << endl;
    else
        cout << "P1 is NOT on zero point" << endl;
    //-----
    P2.move(0, 0); // P2 moves to (0, 0)
    if( P2.is_zero() )
        cout << "P2 is on zero point" << endl;
    else
        cout << "P2 is NOT on zero point" << endl;
}
```



# Defining Methods inside a Class

- Any function (method) code written inside a class definition is considered automatically an **inline function** by the compiler.
- Example: ***is\_zero*** function of Point class can be defined inside the class. The "inline" keyword is not written.
- Inline syntax should be preferred only for short functions.

```
class Point { // Declaration of Point Class
    int x, y; // Properties: x and y coordinates

    public:
        void move(int, int);
        void print();

        bool is_zero() // Function is written inline
        {
            return (x == 0) && (y == 0);
        }

};
```

# Defining Dynamically Allocated Objects

Two pointers (ptr1 and ptr2) to Point objects are defined and dynamically allocated.

```
int main() {  
    // Allocating memory for two Point objects  
    Point * ptr1 = new Point;  
    Point * ptr2 = new Point;  
  
    ptr1->move(50, 50);  
    ptr1->print();  
  
    ptr2->move(100, 150);  
    if( ptr2->is_zero() )  
        cout << " Object is on zero." << endl;  
    else  
        cout << " Object is NOT on zero." << endl;  
  
    // Releasing the memory  
    delete ptr1;  
    delete ptr2;  
}
```

# Defining Array of Objects

An array is defined with 10 elements of type Point.

```
int main() {  
    Point array[10];    // defining an array with 10 objects  
  
    array[0].move(15, 40);  
    // move message to the first element (index 0)  
  
    array[1].move(75, 35);  
    // move message to the second element (index 1)  
  
    for (int i = 0; i < 10; i++)  
        // print message to all objects in the array  
        array[i].print();  
}
```

# Controlling Access to Members

- The access specifiers **public:** , **private:** , **protected:** are used to control access to member data and functions of a class.
- Each access specifier applies until the next access specifier, or until the end of class declaration.
- **Public** members may be accessed from any place in the program.
- **Private** members can be accessed only by member functions of that class. It is the **default** access mode.
- Private data are hidden, and can not be used by main program or by other classes.
- **Protected** is similar to private, and related to inheritance. It allows derived class to access member data and functions of base class.

# Example: Controlling Access to Members

- Only the **public** members (data and function) of the class A can be accessed from outside of class such as in main program.
- The f1,f2,f3 functions can access to all member data (x,y,z).

```
#include <iostream>
using namespace std;

class A
{
    // Member data
    private    : int x;
    protected : int y;
    public     : int z;

    // Member functions
    private    : void f1() {}
    protected : void f2() {}
    public     : void f3() {}
};
```

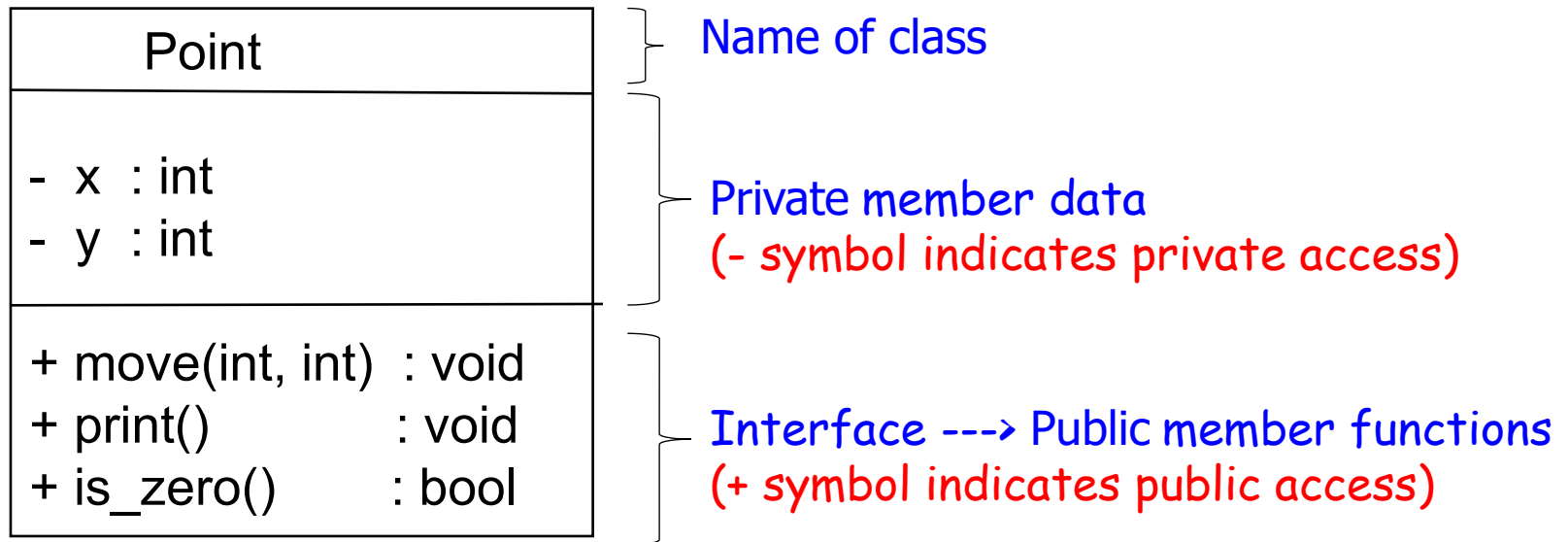
```
int main()
{
    A a;    // Object definition (variable)

    a.x = 10; // Compiler error , x is private
    a.y = 20; // Compiler error , y is protected
    a.z = 30; // z is public

    a.f1(); // Compiler error , f1 is private
    a.f2(); // Compiler error , f2 is protected
    a.f3(); // f3 is public
}
```

# UML class diagram for Point class

- The purpose of public members is to present a view of the *services (functions)* the class provides.
- This set of services forms the *public interface* of the class.
- The private members are not accessible to the clients of a class.



# Example: bool as Function return type

In the following version of Point class, the return type of the **move** function is changed from **void** to **bool**.

```
class Point
{
    private:
        int x, y; // private members

    public: // public members
        bool move(int, int);
        void print();
        bool is_zero();
};
```

Suppose that clients (main program or other classes) of the Point class are not allowed to move a point object outside a graphics window with a size of 500x300 pixels.

```
// A function to move the points  
  
bool Point :: move (int new_x, int new_y)  
{  
    if ( new_x > 0  &&  new_x < 500  &&  
        new_y > 0  &&  new_y < 300)  
    {  
        // assign new values  
        x = new_x;  
        y = new_y;  
        return true; // input values are accepted  
    }  
  
    return false; // input values are not accepted  
}
```



The **move** function returns a **boolean value** to inform whether the input values are accepted or not.

```
int main()
{
    Point p1;    // p1 object is defined
    int x, y;
    // Two variables to read coordinate values from keyboard

    cout << " Give x and y coordinates ";
    cin >> x >> y;    // Read two values from keyboard

    if ( p1.move(x, y) ) // send move message and check the result
        p1.print();    // If result is true, print coordinates on screen
    else
        cout << endl << "Input values are not accepted";
}
```

# Example : private data members

- If the access specifier is not written for a member data or function, then they are **private by default**.
- In Point class, x and y data members are **private** by default.
- It is not possible to assign a value to x or y directly outside the class.
- Also, displaying them directly is not allowed.

```
class Point {  
    int x, y; // private member data by default  
public:  
    bool move(int, int);  
    void print();  
};  
int main() {  
    Point p1;  
    p1.move(100, 50);  
    p1.print();  
    p1.x = 70;    // Compiler error  
    p1.y = 130;   // Compiler error  
    cout << p1.x << endl; // Compiler error  
    cout << p1.y << endl; // Compiler error  
}
```

# Example : public data members

When x and y members are defined as **public**, there will be no access restrictions.

```
class Point {  
    public:  
        int x, y;  
        bool move(int, int);  
        void print();  
        ...  
};  
  
int main() {  
    Point p1;  
    p1.move(100, 50);  
    p1.print();  
    p1.x = 70;    // Allowed  
    p1.y = 130;   // Allowed  
    cout << p1.x << endl; // Allowed  
    cout << p1.y << endl; // Allowed  
}
```

# Example: private function members

**Private** member functions are not allowed to call directly such as from main.

```
class Point {  
    // The move function is private by default  
    bool move(int, int);  
  
    public:  
        int x, y;  
        void print();  
        ...  
};  
  
int main() {  
    Point p1;  
    p1.move(100, 50); // Compiler error!  
    p1.print();  
    p1.x = 70;  
    p1.y = 130;  
    cout << p1.x << endl;  
    cout << p1.y << endl;  
}
```

# Friend Classes

- A function or an entire class may be declared to be a friend of another class.
- A *friend* of a class has the right to access all members (private, protected or public) of the given class.

```
class A
{
    friend class B;

    private: // private members of A
        int i;

    public: // public members of A
        void func1();
};
```

- Class B is friend of class A.
- Class B can access members of class A.
- But class A can not access members of class B.

Class B can access private members (data & function) of class A.

```
class B
{
    int j;

    public:
        void func2(A x)
        //Takes an object of class A, as argument
        {
            cout << j << endl;
            cout << x.i; // i is member of class A
            x.func1();    // func1 is member of class A
        }
};
```

The object **a** is passed to member function of the object **b**.

```
int main() {
    A a;
    B b;
    b.func2 ( a );
}
```

# Friend Functions

- A **friend function** is **not a member** of any class.
- A *friend* function has the right to access all members (private, protected or public) of the specified class.
- If a member (data or function) of a class is already defined as public, then it is not necessary to declare any other class or function as friend.

```
class Point
{
    // Define a friend function of the Point class
    friend void set_to_zero (Point &);

    int x, y;    // private members: x and y coordinates

    public:    // public members
        bool move(int, int);
        void print();
        bool is_zero();
};
```

The **set\_to\_zero** friend function can access private members of the Point class.

```
// Non-member function  
// Not a member of any class  
  
void set_to_zero (Point &p)  
{  
    p.x = 0;  
    p.y = 0;  
}
```

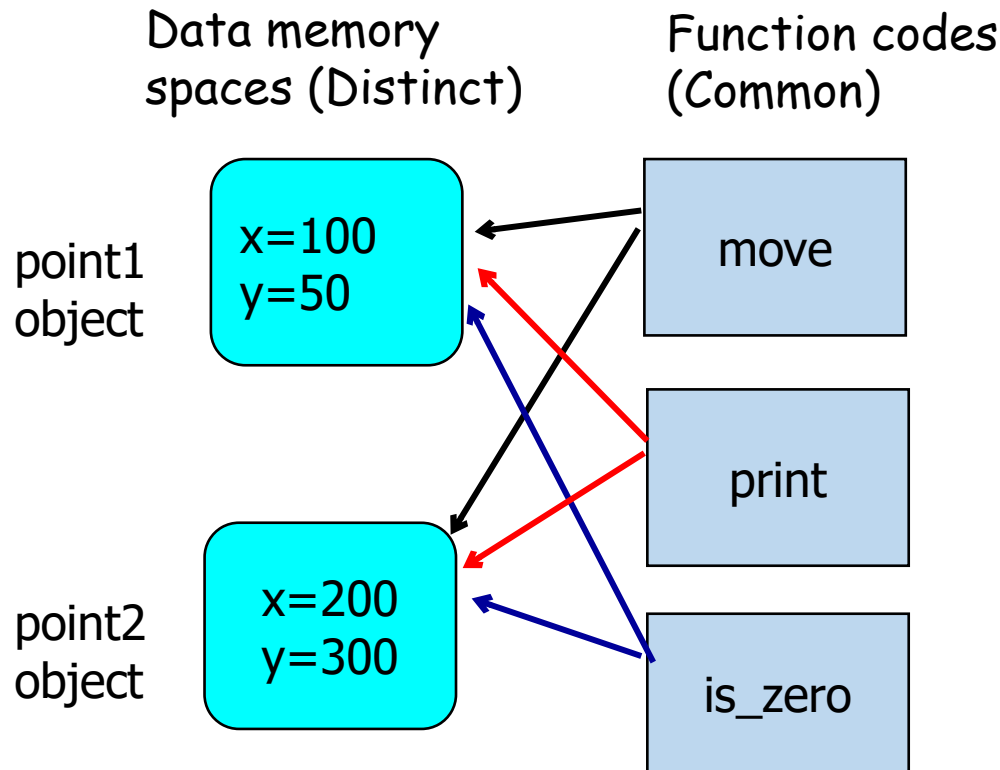
The object p1 is passed to the **set\_to\_zero** friend function.

```
int main()  
{  
    Point p1;  
    set_to_zero(p1);  
}
```



# Data memory spaces and Function codes

- Each object has its own distinct data space in memory.
- When an object is defined, memory is allocated only for its data members.
- Each object of the same class uses the same common function code.



# The **this** Pointer

- For every class definition, C++ compiler maintains a special built-in pointer, called the **this** pointer.
- When a member function is called, the **this** pointer contains the **self address** of the object.
- **Example:** point1 and point2 objects have different **this** pointers.
- The member functions of Point class can access data members using the **this** pointer.

In the following examples, usage of the **this** pointer is optional (**not required**).


```
void Point::move(int new_x, int new_y)
{
    // assigns new values to coordinates
    this->x = new_x;
    this->y = new_y;
}
```

```
void Point::print()
{
    cout << this->x << " " << this->y << endl;
}
```

# Example: Member data and Argument with same name

When arguments (parameters) of a function has the **same names** as the data members of class, the **this** pointer is **required** to be used in the function.

```
class Point {  
    int x, y;           // private members  
    public:              // public members  
        bool move (int x, int y);  
        .....  
};
```



```
bool Point :: move (int x, int y) // parameters has the same name as  
{                                  // data members x and y  
    if( x > 0 && x < 500 && y > 0 && y < 300) {  
        this->x = x; // assigns given x value to member x  
        this->y = y; // assigns given y value to member y  
        return true; // input values are accepted  
    }  
    return false; // input values are not accepted  
}
```

# Example : Function returning an object

- The ***find\_max\_distance()*** member function below takes a Point object as argument.
- It compares distance of itself and the distance of the argument object.
- Then it returns the address of the object, which has the largest distance from the origin.

```
Point * Point :: find_max_distance (Point & p)
{
    int distance1, distance2;

    distance1 = sqrt ( ( x * x) + (y * y) );           // Hypothenus formula
    distance2 = sqrt ( ( p.x * p.x) + (p.y * p.y) ); // Hypothenus formula

    if (distance1 > distance2 )
        return this; // Object returns its own address.
    else
        return &p; // Else returns the address of the p object.
}
```

# Calling the member function of an object

```
int main()
{
    Point P1, P2;    // Two objects: P1 and P2

    P1.move(100, 50);
    P2.move(20, 65);

    Point * a;      // a is a pointer
    a = P1. find_max_distance(P2);

    // Alternative command
    // a = P2. find_max_distance(P1);

    // The point that has the largest distance
    // is printed on screen
    a->print();
}
```

# Alternative: Calling a non-member function

```
class Point {  
public:  
    int x, y; // Data members are public  
    .....  
};
```

```
// Non-member function  
Point * find_max_distance (Point & p1, Point & p2) {  
    int distance1 = sqrt ( ( p1.x * p1.x) + (p1.y * p1.y) );  
    int distance2 = sqrt ( ( p2.x * p2.x) + (p2.y * p2.y) );  
    if (distance1 > distance2 ) return &p1;  
    else return &p2;  
}
```

```
int main() {  
    Point P1, P2;  
    P1.move(100, 50);  
    P2.move(20, 65);  
    Point * a;  
    a = find_max_distance(P1, P2);  
    a->print();  
}
```