

# GE3 Computer Science

C and C ++ Lecture series *for*  
B.SC 3<sup>rd</sup> semester *by*

**Subhadip Mukherjee**

Department of computer science

Kharagpur College

**LECTURE 14**

# Object-Oriented Programming

## Introduction to Classes

- Class Definition
- Class Examples
- Objects
- Constructors
- Destructors

# Class

- The class is the **cornerstone** of C++
  - It makes possible encapsulation, data hiding and inheritance
- **Type**
  - Concrete representation of a concept
    - Eg. **float** with operations like -, \*, + (math real numbers)
- **Class**
  - A user defined type
  - Consists of both data and methods
  - Defines properties and behavior of that type
- **Advantages**
  - Types matching program concepts
    - Game Program (Explosion type)
  - Concise program
  - Code analysis easy
  - Compiler can detect illegal uses of types
- **Data Abstraction**
  - Separate the implementation details from its essential properties

# Classes & Objects

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

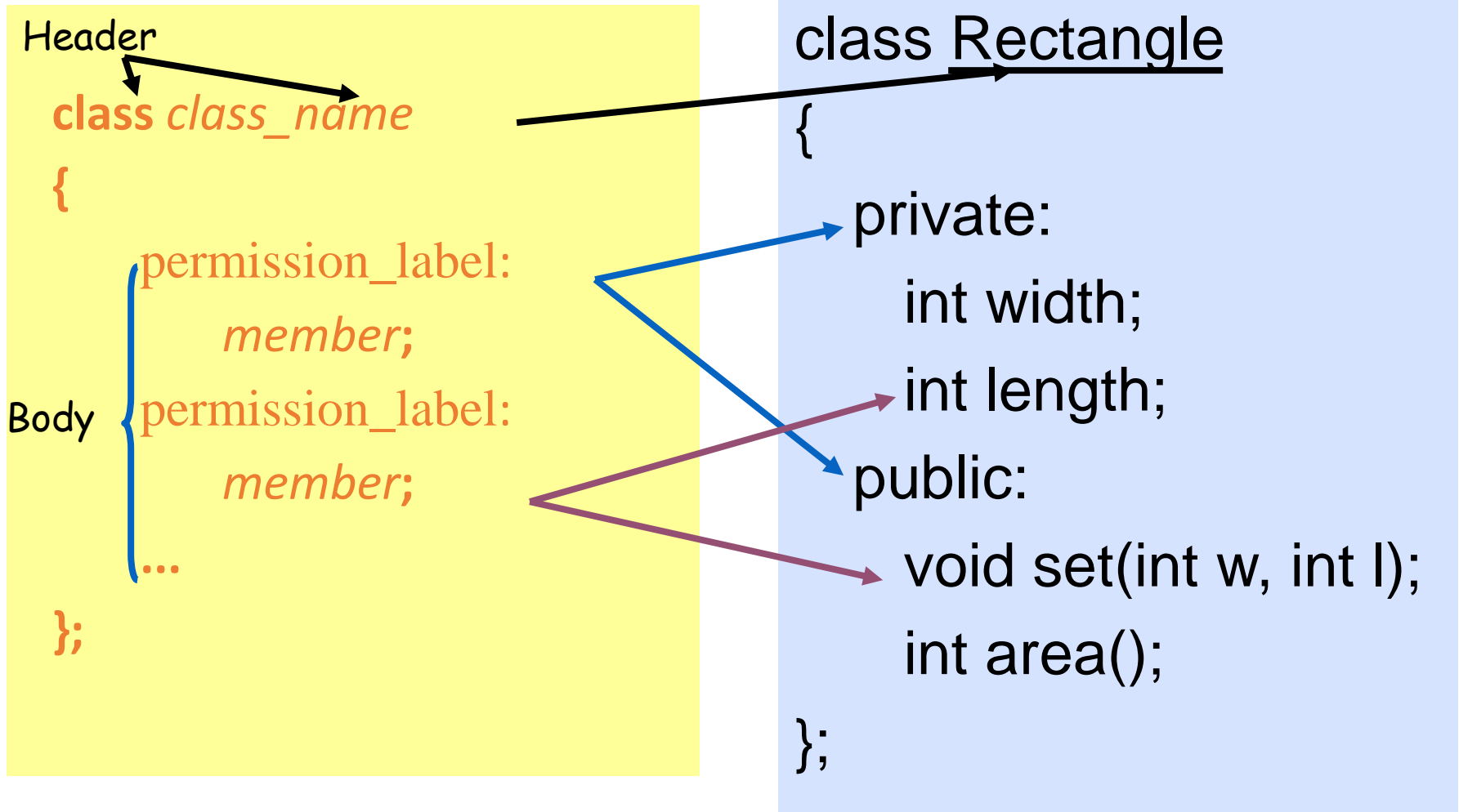
**Objects:** Instance of a class

```
Rectangle r1;
Rectangle r2;
Rectangle r3;
```

⋮

```
int a;
```

# Define a Class Type



# Class Definition

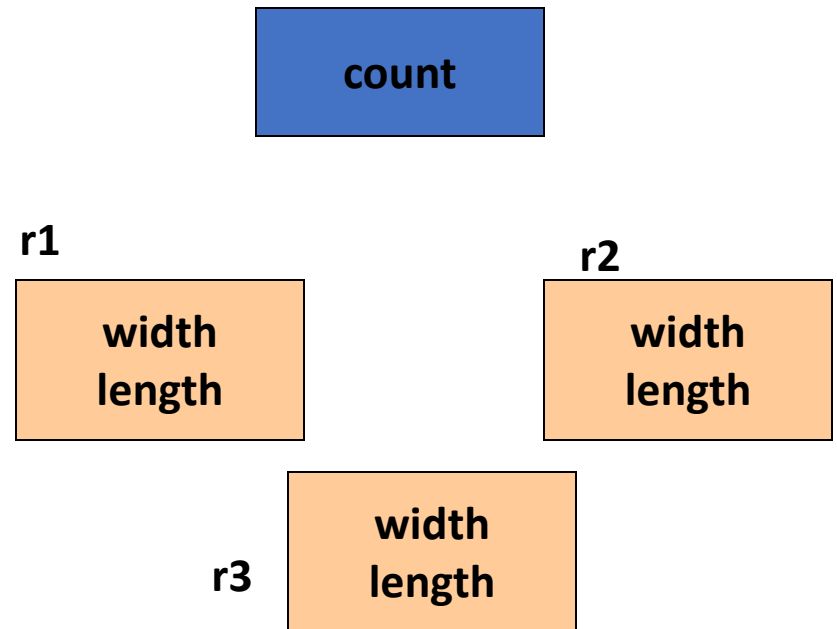
## Data Members

- Can be of any type, built-in or user-defined
- *non-static data member*
  - Each class object has its own copy
- *static data member*
  - Acts as a global variable
  - One copy per class type, e.g. counter

# Static Data Member

```
class Rectangle
{
    private:
        int width;
        int length;
    → static int count;
    public:
        void set(int w, int l);
        int area();
}
```

```
Rectangle r1;
Rectangle r2;
Rectangle r3;
```



# Class Definition

## Member Functions

- Used to
  - access the values of the data members (**accessor**)
  - perform operations on the data members (**implementor**)
- Are declared inside the class body
- Their definition can be placed inside the class body, or outside the class body
- Can access both public and private members of the class
- Can be referred to using dot or arrow member access operator



# Define a Member Function

```
class Rectangle
{
  private:
    int width, length;
  public:
    void set (int w, int l);
    int area() {return width*length; }
};
```

class name

member function name

inline

```
r1.set(5,8);
rp->set(8,10);
```

```
void Rectangle :: set (int w, int l)
{
  width = w;
  length = l;
}
```

scope operator

# Class Definition

## Member Functions

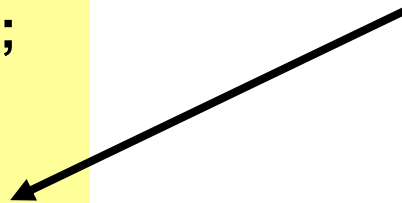
- **const** member function
  - **declaration**
    - *return\_type func\_name (para\_list) const;*
  - **definition**
    - *return\_type func\_name (para\_list) const { ... }*
    - *return\_type class\_name :: func\_name (para\_list) const { ... }*
  - **Makes no modification about the data members (safe function)**
  - **It is illegal for a const member function to modify a class data member**

# Const Member Function

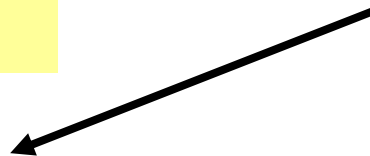
```
class Time
{
private :
    int  hrs, mins, secs ;

public :
    void  Write ( ) const ;
};
```

function declaration



function definition



```
void Time :: Write( ) const
{
    cout <<hrs << ":" << mins << ":" << secs << endl;
}
```

# Class Definition - Access Control

- **Information hiding**
  - To prevent the internal representation from direct access from outside the class
- **Access Specifiers**
  - **public**
    - may be accessible from anywhere within a program
  - **private**
    - may be accessed only by the member functions, and friends of this class
  - **protected**
    - acts as public for derived classes
    - behaves as private for the rest of the program

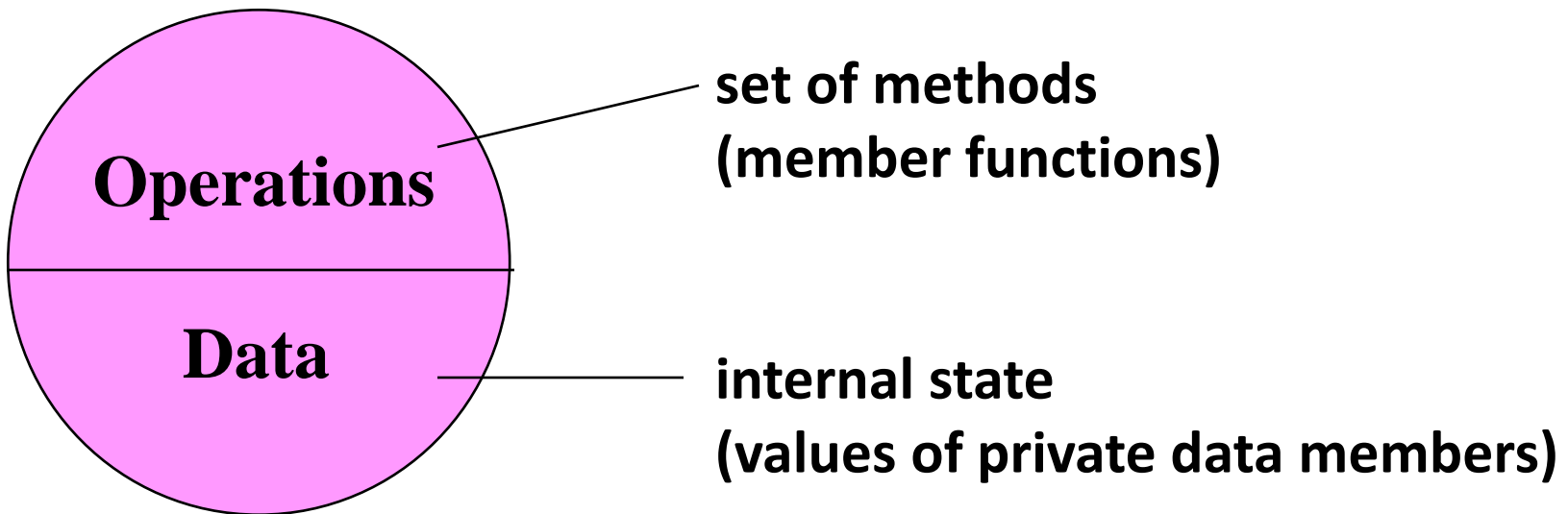
# Class Definition

## Access Control

- The default access specifier is *private*
- The data members are usually private or protected
- A **private** member function is a helper, may only be accessed by another member function of the same class (exception *friend* function)
- The **public** member functions are part of the *class interface*
- Each access control section is optional, repeatable, and sections may occur in any order

# What is an object?

## **OBJECT**



# Declaration of an Object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

```
main()
{
    Rectangle r1;
    Rectangle r2;

    r1.set(5, 8);
    cout<<r1.area()<<endl;

    r2.set(8,10);
    cout<<r2.area()<<endl;
}
```

# Another Example

```
#include <iostream.h>

class circle
{
private:
    double radius;

public:
    void store(double);
    double area(void);
    void display(void);
};
```

```
// member function definitions
```

```
void circle::store(double r)
{
    radius = r;
}
```

```
double circle::area(void)
{
    return 3.14*radius*radius;
}
```

```
void circle::display(void)
{
    cout << "r = " << radius << endl;
}
```

```
int main(void) {
    circle c; // an object of circle class
    c.store(5.0);
    cout << "The area of circle c is " << c.area() << endl;
    c.display();
}
```



# Declaration of an Object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

**r1 is statically allocated**

```
main()
{
    Rectangle r1;
    → r1.set(5, 8);
}
```

**r1**

**width = 5  
length = 8**

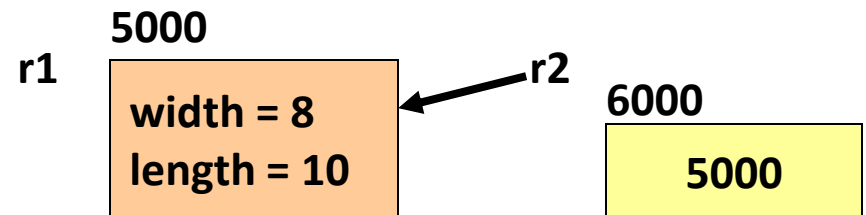
# Declaration of an Object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

r2 is a pointer to a Rectangle object

```
main()
{
    Rectangle r1;
    r1.set(5, 8);    //dot notation

    Rectangle *r2;
    r2 = &r1;
    r2->set(8,10);  //arrow notation
}
```



# Declaration of an Object

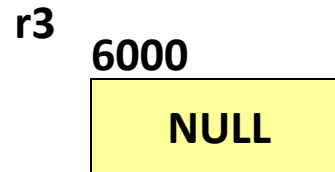
```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

r3 is dynamically allocated

```
main()
{
    Rectangle *r3;
    r3 = new Rectangle();

    r3->set(80,100); //arrow notation

    delete r3;
    → r3 = NULL;
}
```



# Object Initialization

```
#include <iostream.h>

class circle
{
public:
    double radius;
};
```

```
int main()
{
    circle c1;           // Declare an instance of the class circle
    c1.radius = 5;      // Initialize by assignment
}
```

## 1. By Assignment

- Only work for public data members
- No control over the operations on data members

# Object Initialization

```
#include <iostream.h>

class circle
{
private:
    double radius;

public:
    void set (double r)
        {radius = r;}
    double get_r ()
        {return radius;}
};
```

## 2. By Public Member Functions

```
int main(void) {
    circle c;           // an object of circle class
    c.set(5.0);        // initialize an object with a public member function
    cout << "The radius of circle c is " << c.get_r() << endl;
    // access a private data member with an accessor
}
```

# Declaration of an Object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

**r2 is a pointer to a Rectangle object**

```
main()
{
    Rectangle r1;
    r1.set(5, 8);    //dot notation

    Rectangle *r2;
    r2 = &r1;
    r2->set(8,10);  //arrow notation
}
```

**r1 and r2 are both initialized by public member function set**

# Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle();
        Rectangle(const Rectangle &r);
        Rectangle(int w, int l);
        void set(int w, int l);
        int area();
}
```

## 3. By Constructor

- Default constructor
- Copy constructor
- Constructor with parameters

**They are publicly accessible**

**Have the same name as the class**

**There is no return type**

**Are used to initialize class data members**

**They have different signatures**

# Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

When a class is declared with no constructors, the compiler automatically assumes **default** constructor and **copy** constructor for it.

- Default constructor

```
Rectangle :: Rectangle() { };
```

- Copy constructor

```
Rectangle :: Rectangle (const
    Rectangle & r)
{
    width = r.width; length = r.length;
};
```



# Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

- Initialize with default constructor

```
Rectangle r1;
Rectangle *r3 = new Rectangle();
```

- Initialize with **copy** constructor

```
Rectangle r4;
r4.set(60,80);

Rectangle r5 = r4;
Rectangle r6(r4);

Rectangle *r7 = new Rectangle(r4);
```

# Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle(int w, int l)
            { width =w; length=l;}
        void set(int w, int l);
        int area();
}
```

If any constructor with any number of parameters is declared, no **default** constructor will exist, unless you define it.

```
Rectangle r4; // error
```

- Initialize with constructor

```
Rectangle r5(60,80);
```

```
Rectangle *r6 = new Rectangle(60,80);
```

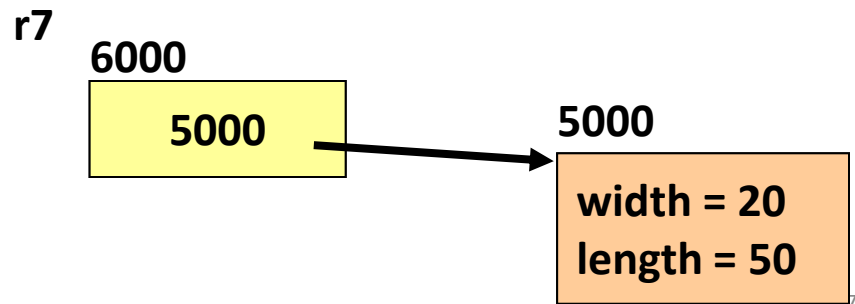
# Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle();
        Rectangle(int w, int l);
        void set(int w, int l);
        int area();
}
```

Write your own constructors

```
Rectangle :: Rectangle()
{
    width = 20;
    length = 50;
};
```

```
Rectangle *r7 = new Rectangle();
```



# Object Initialization

```
class Account
{
    private:
        char *name;
        double balance;
        unsigned int id;
    public:
        Account();
        Account(const Account &a);
        Account(const char *person);
}
```

```
Account :: Account()
{
    name = NULL; balance = 0.0;
    id = 0;
};
```

**With constructors, we have more control over the data members**

```
Account :: Account(const Account &a)
{
    name = new char[strlen(a.name)+1];
    strcpy (name, a.name);
    balance = a.balance;
    id = a.id;
};
```

```
Account :: Account(const char *person)
{
    name = new char[strlen(person)+1];
    strcpy (name, person);
    balance = 0.0;
    id = 0;
};
```

**Thank You**