Object Oriented Programming

CSE 1203

Books and References

- Object Oriented Programming by Balagurusamy
- Object Oriented Programming by Lafore
- Java Programming by Liang
- Online materials

What programming is?

Programming is taking

- A ***problem***
    - Find the area of a rectangle
- A set of ***data***
    - length
    - width
- A set of ***functions***
    - area = length * width

Then

- Applying functions to data to get answer

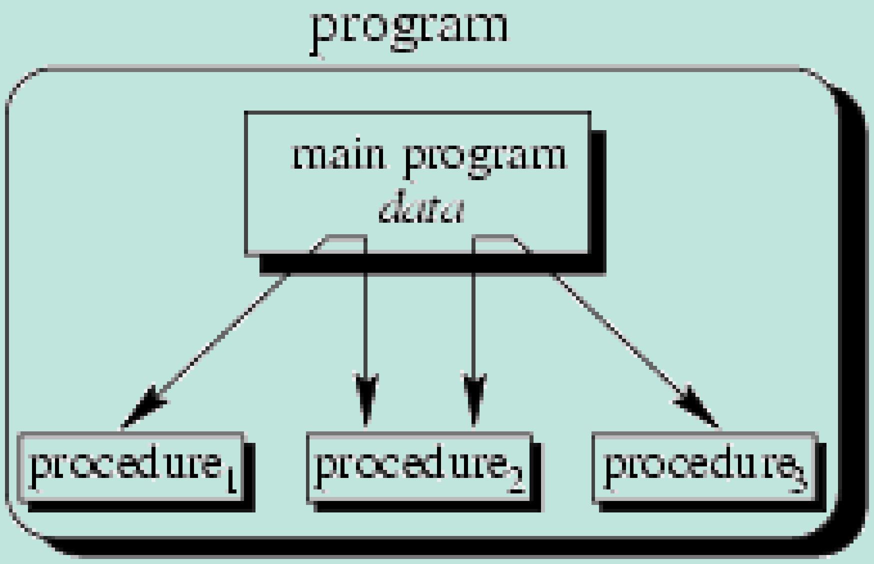
Programming Concept Evolution

- Unstructured
- Procedural
- Object-Oriented

Unstructured Programming

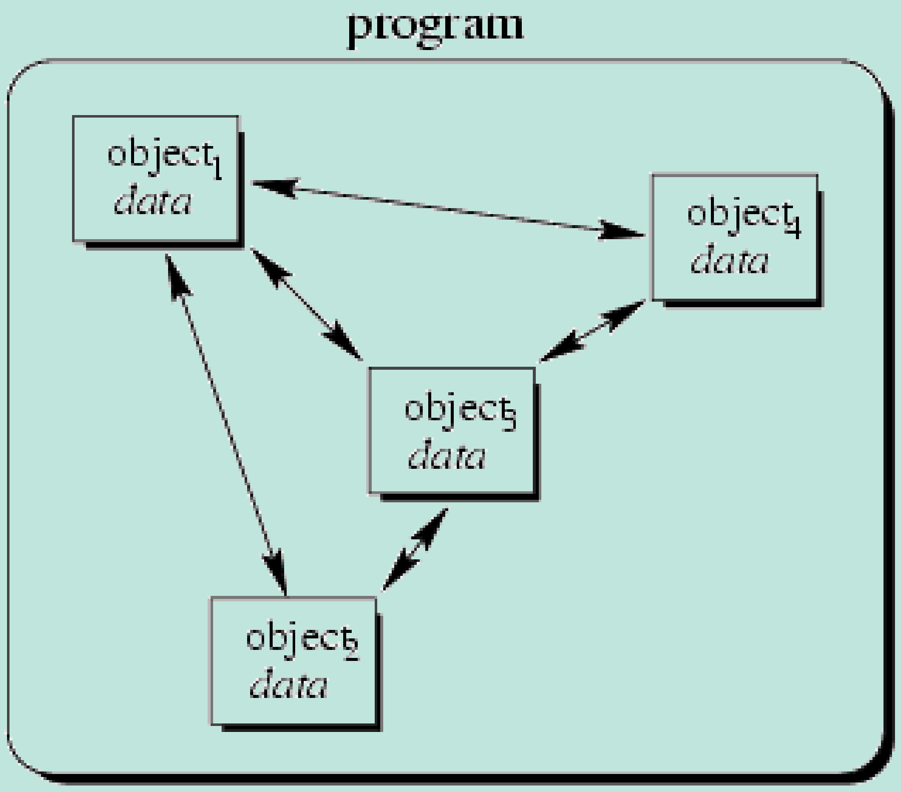
Unstructured Programming is **a type of programming that generally executes in sequential order** i.e., these programs just not jumped from any line of code and each line gets executed sequentially. There are both high- and low-level programming languages that use non-structured programming. Some languages commonly cited as being non-structured include **JOSS, FOCAL, TELCOMP, assembly languages, MS-DOS batch files, and early versions of BASIC, Fortran, COBOL, and MUMPS**.

Procedural Concept



- The main program coordinates calls to procedures and hands over appropriate data as parameters.

Object-Oriented Concept



- Objects of the program interact by sending messages to each other

Objects

An object is an encapsulation of both functions and data

(not one or the other individually)

- ***Objects are an Abstraction***
  - *represent real world entities*
  - *Classes are data type that define shared common properties or attributes*
  - *Objects are instances of a class*
- ***Objects have State***
  - *have a value and a particular time*
- ***Objects have Operations***
  - *associated set of operations called methods that describe how to carry out operations*
- ***Objects have Messages***
  - *request an object to carry out one of its operations by sending it a message*
  - *messages are the means by which we exchange data between objects*

OO Perspective

Let's look at our earlier Rectangle through object oriented eyes:

Object

Rectangle

data - *encapsulated*

width

length

function (called a method)- *encapsulated*

*area = length * width*

In our object oriented program, we will have an instance of the class Rectangle.

If we wish to find the area of the rectangle, we send a request to the object instance telling the rectangle to return its area.

In C++, rather than writing a procedure, we define a class that encapsulates the knowledge necessary to answer the question - here, what is the area of the rectangle.

Example Object Oriented Code

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

```
   int area()
   {
         return width*length;
   }
}
```

```
main()
{
   Rectangle rect(3,5);
    cout<<rect.area()<<endl;
}
```

Object-Oriented Programming Languages

- Characteristics of OOPL:
  - Encapsulation
  - Inheritance
  - Polymorphism

Characteristics of OOPL

- **Encapsulation=**combining data structure with actions

  -actions -> permissible behaviors of objects that are controlled through the member functions

  -data structure -> represents the properties, the state, or characteristics of objects

  -information hiding = process of making certain items inaccessible

- **Inheritance=**ability to derive new objects from old

  -permits objects of a more specific class to inherit the properties (data) and behaviors (functions) of a more general class

  -ability to define a hierarchical relationship between objects

- **Polymorphism=**how objects respond to certain kinds of messages

  -ability for different objects to interpret functions differently

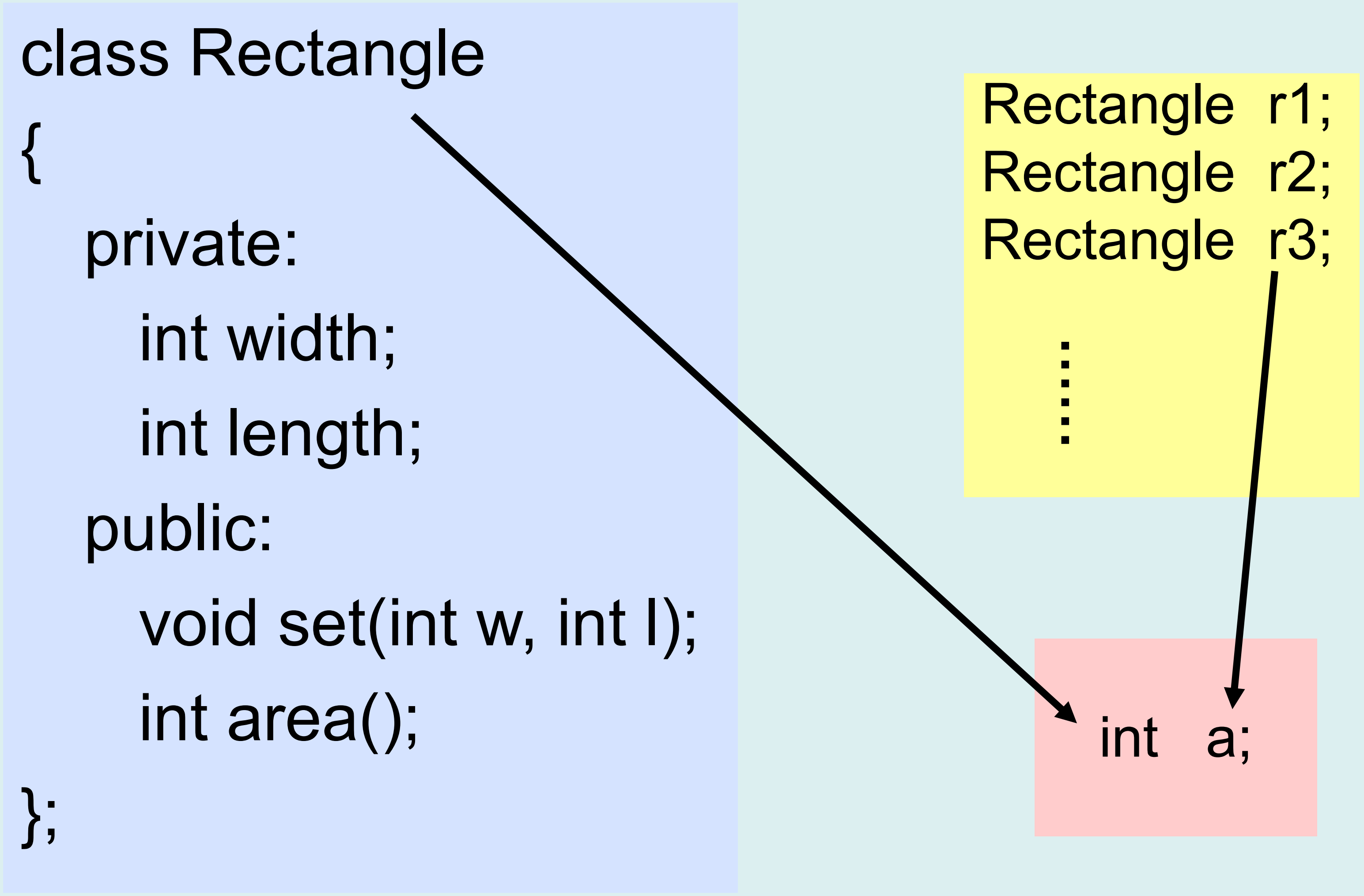
Object-Oriented Programming-- Introduction to Classes

- Class Definition
- Class Examples
- Objects

Classes & Objects

- The class is the cornerstone of C++
  - It gives the C++ its identity from C
  - It makes possible encapsulation, data hiding and inheritance
- Class:
  - Consists of both data and methods
  - Defines properties and behavior of a set of entities
- Object:
  - An instance of a class
  - A variable identified by a unique name

Classes & Objects



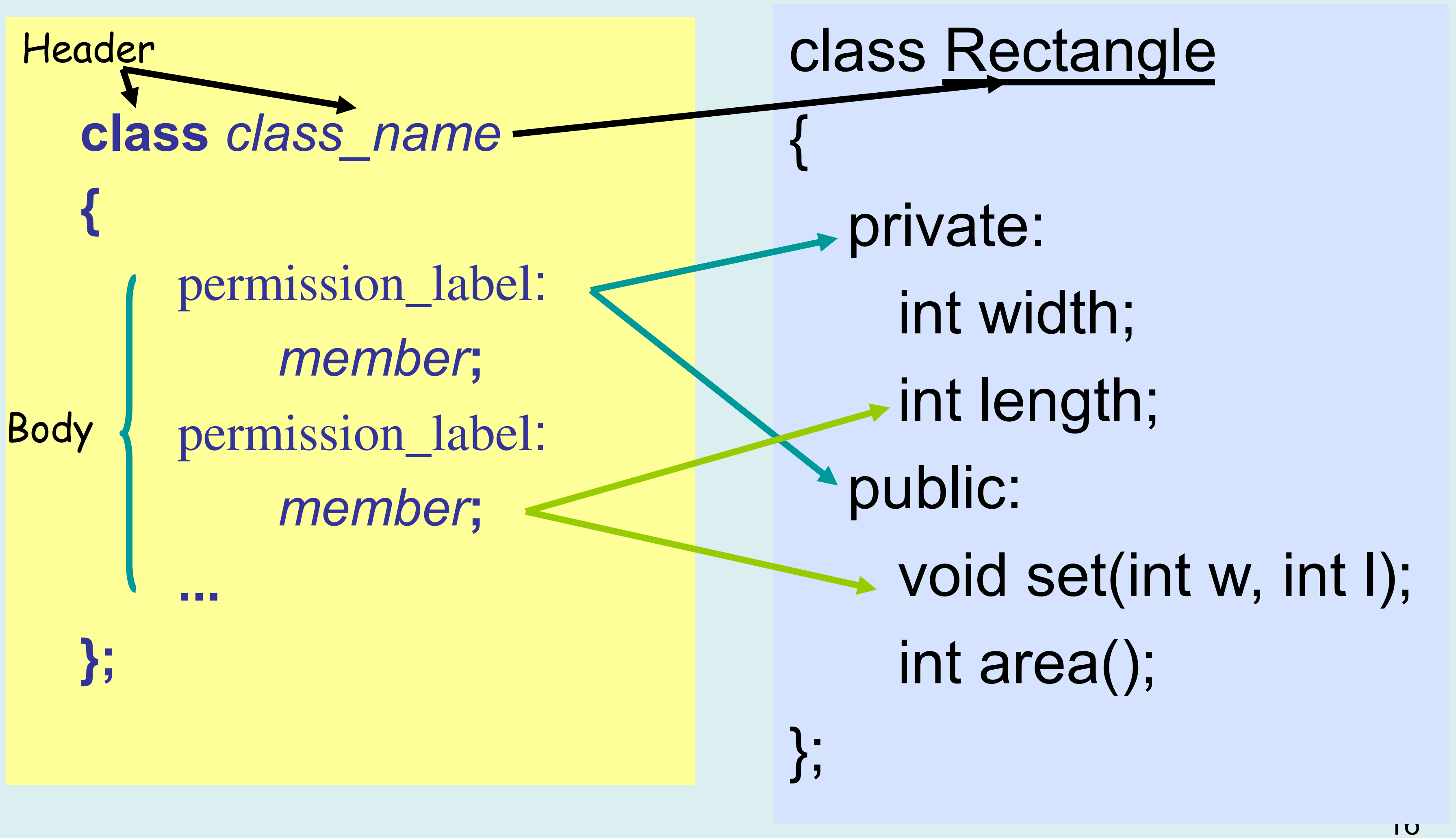
```
class Rectangle
{
   private:
      int width;
      int length;
   public:
      void set(int w, int l);
      int area();
};
```

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

   ……
```

```
int  a;
```

Define a Class Type



```
class class_name
{
    permission_label:
        member;
    permission_label:
        member;
    ...
};
```

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

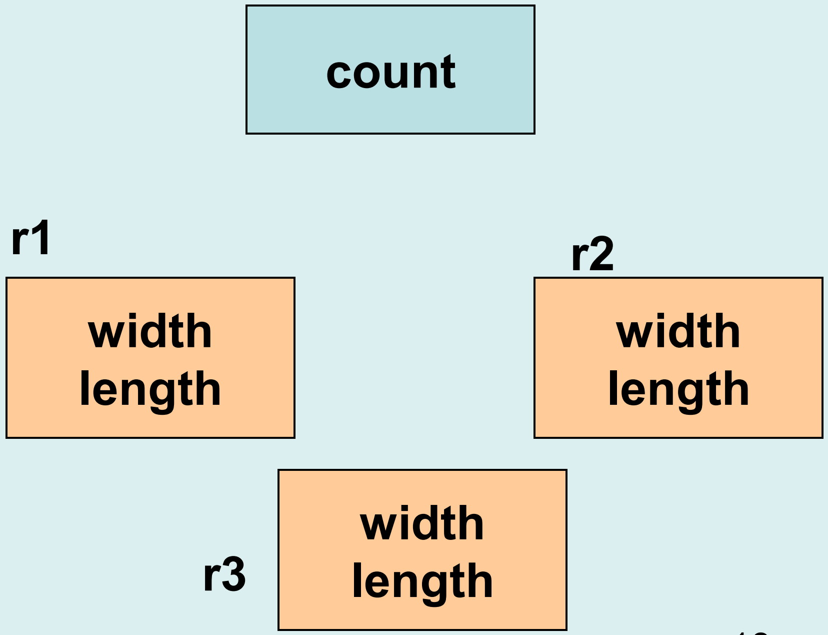
Class Definition-Data Members

- Can be of any type, built-in or user-defined
- *non-static* data member
  - Each class object has its own copy
  - Cannot be initialized explicitly in the class body
  - Can be initialized with member function, or class constructor
- *static* data member
  - Acts as a global object, part of a class, not part of an object of that class
  - One copy per class type, not one copy per object
  - Can be initialized explicitly in the class body

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– 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, in the same way as declaring a function
- 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; }
};
```

inline

```
r1.set(5,8);

rp->set(8,10);
```

class name

member function name

```
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 Base{
 mutable int x;
 public:
     void setX(int a){ x=a;}
     int getX()const {
         x++;
         return x;}

};
```

function declaration

Data Member can't be changed

To make const function executable →making data member mutable

Class Definition – Member Functions

- **static** member function
  - Static member function can contain only static data member
  - Non-static Static member function can contain both static and non-static data member
  - Static function can run using

    <classname>::<static function()

static– Member Functions

```
class Base{
 int x;
 static int y;
 public:
 Base(int X){
  x=X;
  y++;
 }
 static int getY(){ return y;}
};

int Base::y=0;
```

```
int main()
{
  Base c1(10),c2(20);

  cout<<Base::getY();
    return 0;
}
```

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;
}
```

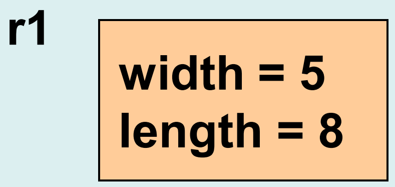
```
int main(void) {
   circle c;   // an object of circle class
   c.store(5.0);
   cout << "The area of circle c is " << c.area() << endl;
   c.display();
}
```

Declaration of an Object

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

```
main()
{
    Rectangle r1;

➡  r1.set(5, 8);
}
```



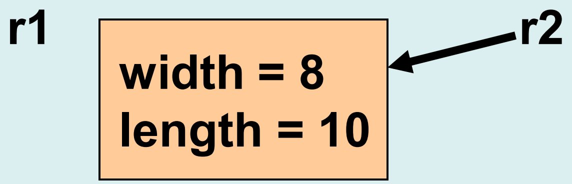
Declaration of an Object

```
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 in heap

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

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

      delete r3;
      r3 = NULL;
}
```

r3

6000

NULL

Object Initialization

1. By Assignment

```
#include <iostream.h>

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

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

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

}
```

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

- Accessor
- Implementor

```
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
}
```

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;
};
```

Copy Constructor

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

Copy member data of one object to another

```
int main(){
 Rectangle r1(2,5);
 Rectangle r2(r1);

}
```

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);
```

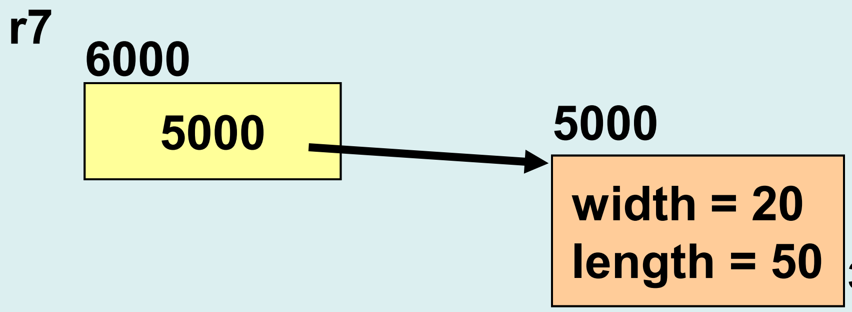
Object Initialization

Write your own constructors

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

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

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



Cleanup of An Object

Destructor

```
class Account
{
    private:
      char *name;
      double balance;
      unsigned int id;  //unique
    public:
      Account();
            ~Account();
}
```

- Its name is the class name preceded by a ~ (tilde)
- It has no argument
- It is used to release dynamically allocated memory and to perform other "cleanup" activities
- It is executed automatically when the object goes out of scope