Object Oriented Programming (CS1143)

Week 4

Department of Computer Science

Capital University of Science and Technology (CUST)

Outline

- Parameter Constructor
- Copy Constructors
- Instance Members
- Class Members

Constructors

- We discussed no parameter constructor (default) last week.
  - Circle ();
- We can have two other types of constructors called
  - Parameter Constructor
  - Copy Constructor
- The following figure shows the declaration of a parameter constructor and a copy constructor along with a default constructor

```
class Circle
{
   ...
   public:
        Circle (double radius);                 // Parameter Constructor
        Circle ();                              // Default Constructor
        Circle (const Circle& circle);          // Copy Constructor
        ...
}
```

Parameter Constructor

- A parameter constructor initializes the data members of each instance with specified values.
- The parameter constructor can be overloaded, which means that we can have several parameter constructors each with a different signature (different parameters).
- The advantage of the parameter constructor is that we can initialize the data members of each object with a specific value.
- For example, if we use a parameter constructor, the radius of one circle can be initialized to 3.1, another one to 4.6, and so on.

The parameter constructor can be overloaded for a class.

Copy Constructor

- Sometimes we want to initialize each data member of an object to the same value as a corresponding data member of a previously created object.
- We can use a copy constructor in this case.
- After calling the copy constructor, the source and the destination objects have exactly the same value for each data member.
- The copy constructor has only one parameter that receives the source object **by reference**.
- We cannot overload the copy constructor because the parameter list is fixed and we cannot have an alternative form.

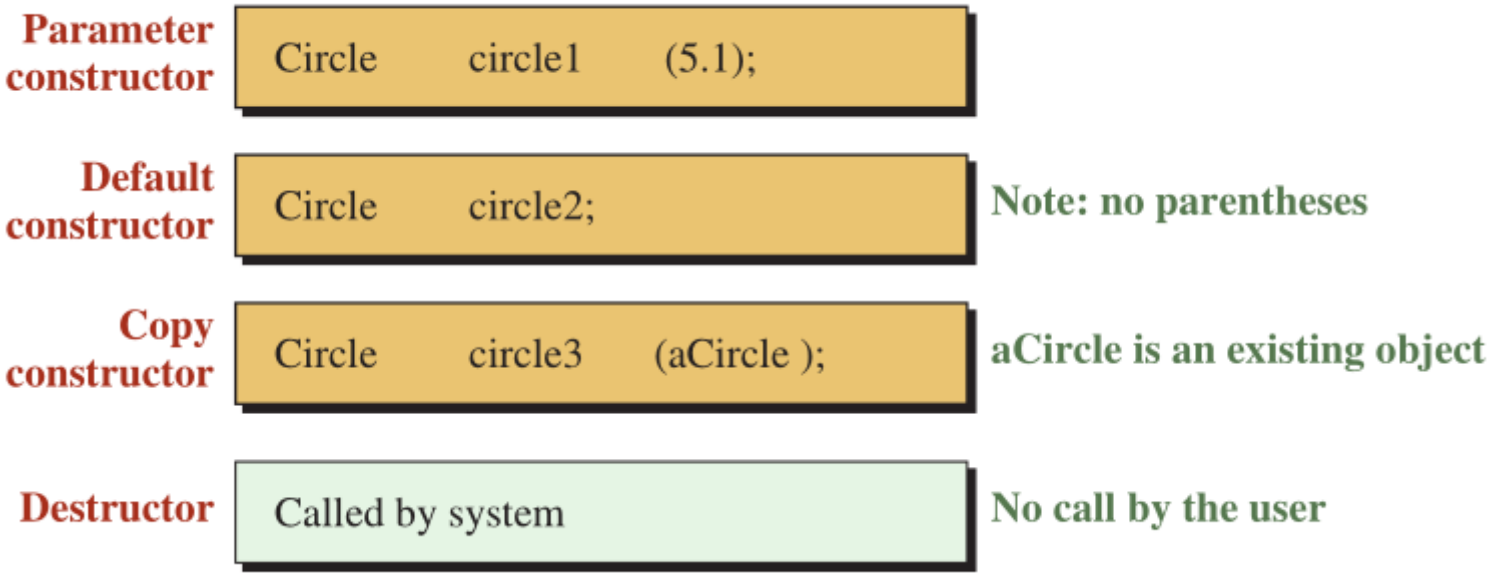
The copy constructor cannot be overloaded for a class.

Definition of a Parameter and Copy Constructor

```
// Definition of a parameter constructor
Circle :: Circle (double rds)
: radius (rds)          // Initialization list
{
   // Any other statements
}
```

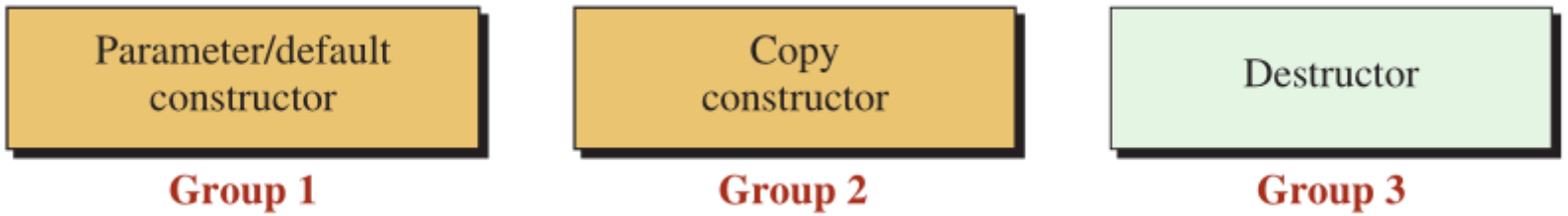
```
// Definition of a copy constructor
Circle :: Circle (const Circle& cr)
: radius (cr.radius)     // Initialization list
{
   // Any other statements
}
```

Object Construction and Destruction



Required Member Functions

- What happens if we ignore declaring and defining one or more of them as we did in the case of our Circle class?
- We put these member functions into three groups.
- We need at least one member from each group. If we do not define at least one member from each group, the system provides one.



A Complete Program

```
1  /*************************************************************
2   * A program to use a class in object-oriented programming   *
3   *************************************************************/
4  #include <iostream>
5  using namespace std;
6
7  /*************************************************************
8   * Class Definition:                                         *
9   * declaration of parameter constructor, default constructor,*
10  * copy constructor, destructor, and other member functions  *
11  *************************************************************/
12 class Circle
13 {
14    private:
15          double radius;
16    public:
17          Circle (double radius);      // Parameter Constructor
18          Circle ();                   // Default Constructor
19          ~Circle ();                  // Destructor
20          Circle (const Circle& circle);      // Copy Constructor
21          void setRadius (double radius);     // Mutator
22          double getRadius () const;          // Accessor
23          double getArea () const;            // Accessor
24          double getPerimeter () const;       // Accessor
25 };
```

```
26  /*****************************************************************
27    * Member Function Definition:                                   *
28    * Definition of parameter constructor, default constructor,     *
29    * copy constructor, destructor, and other member functions      *
30    *****************************************************************/
31  // Definition of parameter constructor
32  Circle :: Circle (double rds)
33  : radius (rds)
34  {
35     cout << "The parameter constructor was called. " << endl;
36  }
37  // Definition of default constructor
38  Circle :: Circle ()
39  : radius (0.0)
40  {
41     cout << "The default constructor was called. " << endl;
42  }
43  // Definition of copy constructor
44  Circle :: Circle (const Circle& circle)
45  : radius (circle.radius)
46  {
47     cout << "The copy constructor was called. " << endl;
48  }
49  // Definition of destructor
50  Circle :: ~Circle ()
51  {
52     cout << "The destructor was called for circle with radius " ;
53     cout << endl;
54  }
```

```
55  // Definition of setRadius member function
56  void Circle :: setRadius (double value)
57  {
58     radius = value;
59  }
60  // Definition of getRadius member function
61  double Circle :: getRadius () const
62  {
63     return radius;
64  }
65  // Definition of getArea member function
66  double Circle :: getArea () const
67  {
68     const double PI  = 3.14;
69     return (PI * radius * radius);
70  }
71  // Definition of getPerimeter member function
72  double Circle :: getPerimeter () const
73  {
74     const double PI  = 3.14;
75     return (2 * PI * radius);
76  }
```

```
77  /*****************************************************************
78   * Application :                                                 *
79   * Creating three objects of class Circle (circle1, circle2,     *
80   * and circle3) and applying some operation on each object       *
81   *****************************************************************/
82  int main ( )
83  {
84     // Instantiation of circle1 and applying operations on it
85     Circle circle1 (5.2);
86     cout << "Radius: " << circle1.getRadius() << endl;
87     cout << "Area: " << circle1.getArea() << endl;
88     cout << "Perimeter: " << circle1.getPerimeter() << endl << endl;
89     // Instantiation of circle2 and applying operations on it
90     Circle circle2 (circle1);
91     cout << "Radius: " << circle2.getRadius() << endl;
92     cout << "Area: " <<  circle2.getArea() << endl;
93     cout << "Perimeter: " << circle2.getPerimeter() << endl << endl;
94     // Instantiation of circle3 and applying operations on it
95     Circle circle3;
96     cout << "Radius: " << circle3.getRadius() << endl;
97     cout << "Area: " <<  circle3.getArea() << endl;
98     cout << "Perimeter: " << circle3.getPerimeter() << endl << endl;
99     // Calls to destructors occur here
100    return 0;
101 }
```

Output

```
The parameter constructor was called.
Radius: 5.2
Area: 84.9056
Perimeter: 32.656

The copy constructor was called.
Radius: 5.2
Area: 84.9056
Perimeter: 32.656

The default constructor was called.
Radius: 0
Area: 0
Perimeter: 0

The destructor was called for circle with radius: 0
The destructor was called for circle with radius: 5.2
The destructor was called for circle with radius: 5.2
```

Description

- The application creates three objects, circle1, circle2, and circle3, using a parameter constructor, a copy constructor, and a default constructor.
- Note that the application does not call the destructor but the system calls it when the object goes out of scope.
- The interesting point is that the objects are destroyed in the reverse order in which they are constructed.
- The last created object is destroyed first; the first created object is destroyed last.
- This is because the objects are created in stack memory. In a stack, the last item inserted is the first item that can be removed.

Instance Members and Class Members

Instance Members and Class Members

- When we design a class, we can have two groups of members:
  - instance members
    - Instance Data Members
    - Instance Member Functions
  - class members (static members)
    - Class Data Members
    - Class Member Functions

Instance Data Members

Instance Data Members

- An instance data member defines the attributes of an instance (object)
- These data members belong exclusively to the corresponding instance and cannot be accessed by other instances.
- Separate regions of memory are assigned for each object and each region stores possibly different values for each data member.
  - Encapsulation

Access Modifier for Instance Data Member

- It makes more sense for instance data members to be private.
- If we make the data members of an instance member public, they can be directly accessed by the application without calling an instance member function.
- In object-oriented programming we want the objects to apply their behaviors on their attributes.
- We must make the instance data members private so that they can be accessed only through instance member functions.

Instance data members encapsulaed in objects

Instance Member Functions

Instance Member Functions

- An instance member function defines one of the behaviors that can be applied on the instance data members of an object.
- There is only one copy of each instance member function in memory and it must be shared by all instances.

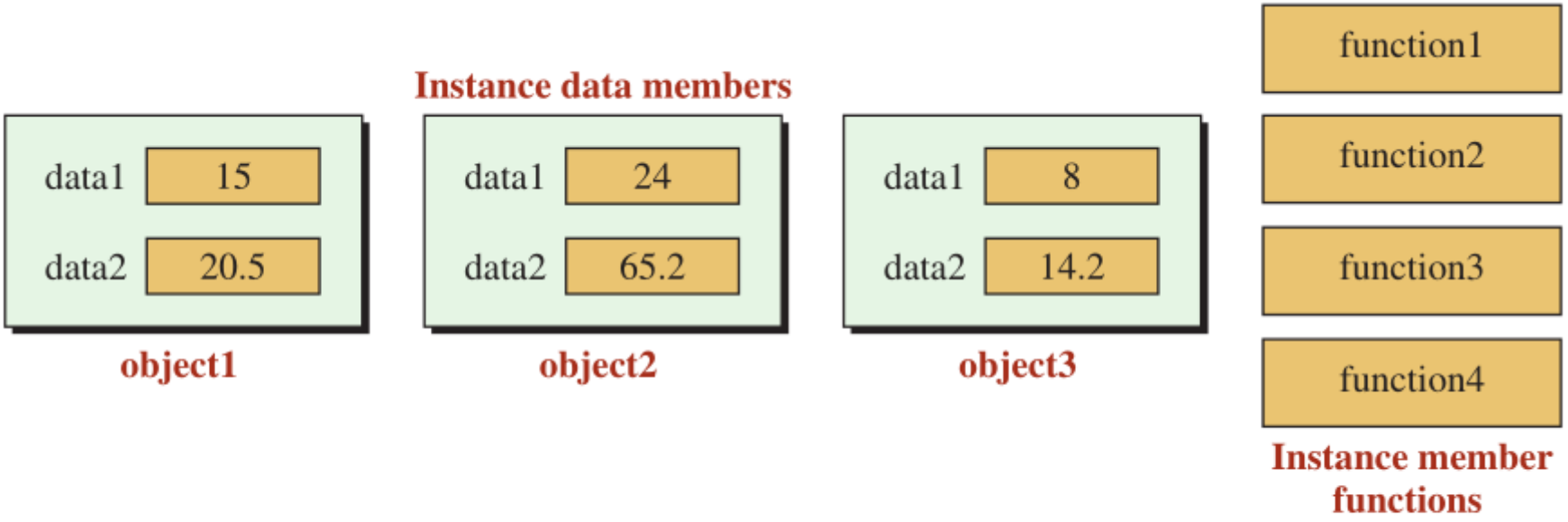


Figure 7.10 A class with two data members and four member functions

Access Modifier for Instance Member Functions

- The access modifier for an instance member function is normally public and allows access from outside the class (the application)
- If the member function is supposed to be used only by other instance member functions within the class, we can make it private.

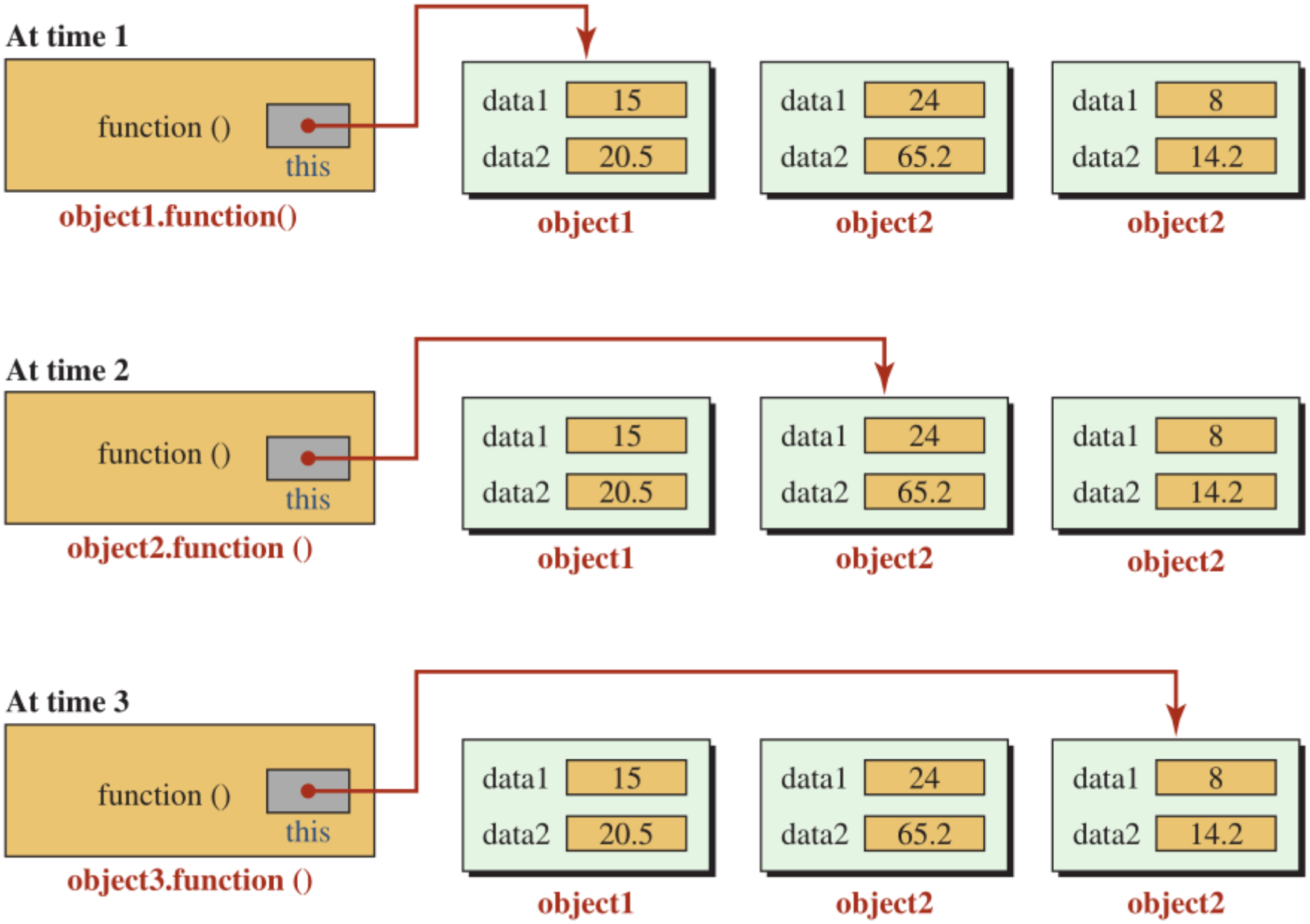
Instance Member Function Selectors

- An application (for example, the main function) can call an instance member function to operate on an instance.
- This call must be done through the instance.
- The application must first construct an instance and then let the instance call the instance member function.
- The C++ language defines two operators, called member selector operators, for this purpose.

Operator	Expression
.	object.member
–>	pointer –> member

this pointer

- If there is only one copy of a member function, how can that function be used by one object at one time and by another object at another time?
- C++ adds a pointer (a variable that holds the address of an object) to each member function.
- The function code is applied to the data members of the object pointed to by the this pointer.


At time 1
function ()
this
object1.function()
data1
15
data2
20.5
object1
data1
24
data2
65.2
object2
data1
8
data2
14.2
object2
At time 2
function ()
this
object2.function ()
data1
15
data2
20.5
object1
data1
24
data2
65.2
object2
data1
8
data2
14.2
object2
At time 3
function ()
this
object3.function ()
data1
15
data2
20.5
object1
data1
24
data2
65.2
object2
data1
8
data2
14.2
object2

Hidden Parameter

- How does an instance member function get a this pointer?
- It is added as a parameter to the instance member function by the compiler as shown below:

```
// Written by the user
double getRadius () const
{
   return radius;
}
```

```
// Changed by the compiler
double getRadius (Circle* this) const
{
   return (this -> radius);
}
```

```
this -> radius    is the same as    (*this).radius
```

```
// Written by the user
circle1.getRadius();
```

```
// Changed by the system
this = &circle1;
getRadius (this);
```

Explicit Use of this Pointer

- We can use the “this” pointer in our program to refer to a data member
- In this way, we do not have to use abbreviated names as we did in the past.
- The this pointer cannot be used in the initialization list of a constructor because at that point, the host object has not been constructed; however, it can be used in the body of the constructor if needed.

```
// Without using the this pointer
void Circle :: setRadius (double rds)
{
   radius = rds;
}
```

```
// Using the this pointer
void Circle :: setRadius (double radius)
{
   this -> radius = radius;
}
```

Class Data Member

Will Study in Week 13