Objects and Classes

Ho Dac Hung

Classes

Objects

- An object is said to be an instance of a class.

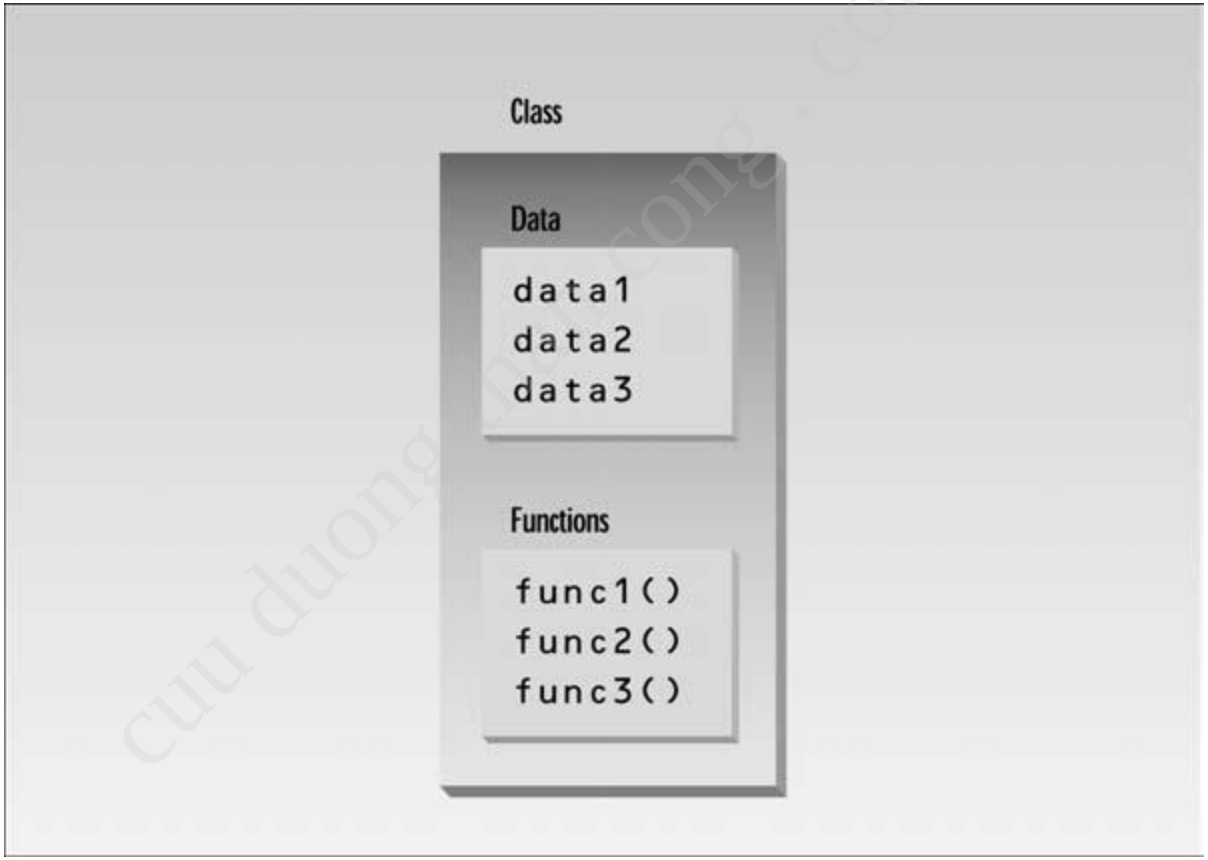
A Simple Class

```
class smallobj //define a class
{
    private:
        int somedata; //class data
    public:
        void setdata(int d)
        { somedata = d; }
        void showdata()
        { cout << “Data is “ << somedata << endl; }
};
```

A Simple Class

```
int main()
{
    smallobj s1, s2;
    s1.setdata(1066);
    s2.setdata(1776);
    s1.showdata();
    s2.showdata();
    return 0;
}
```

A Simple Class



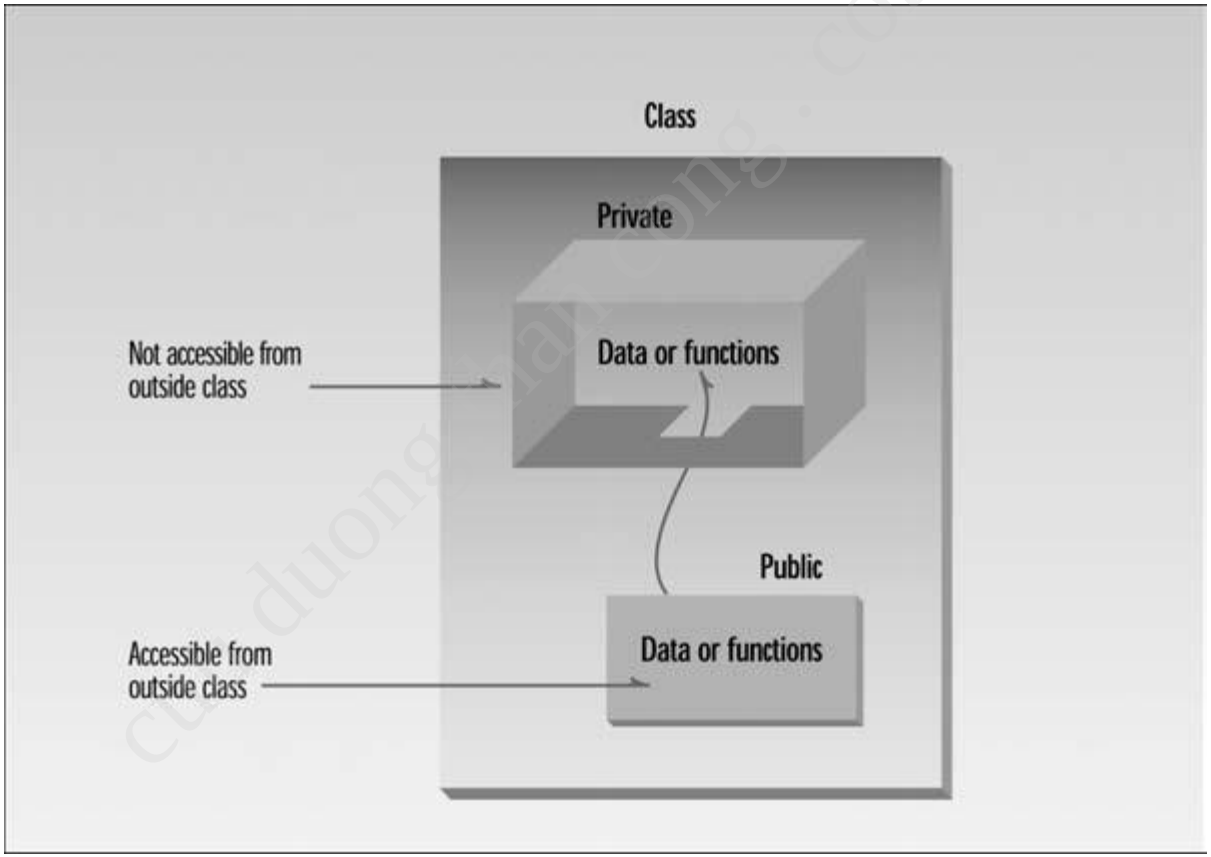
Defining the Class

- The definition starts with the keyword class, followed by the class name.
- Like a structure, the body of the class is delimited by braces and terminated by a semicolon.

Data Hiding

- A key feature of object-oriented programming is data hiding. This term does not refer to the activities of particularly paranoid programmers; rather it means that data is concealed within a class so that it cannot be accessed mistakenly by functions outside the class.
- The primary mechanism for hiding data is to put it in a class and make it private. Private data or functions can only be accessed from within the class. Public data or functions, on the other hand, are accessible from outside the class.

Data Hiding



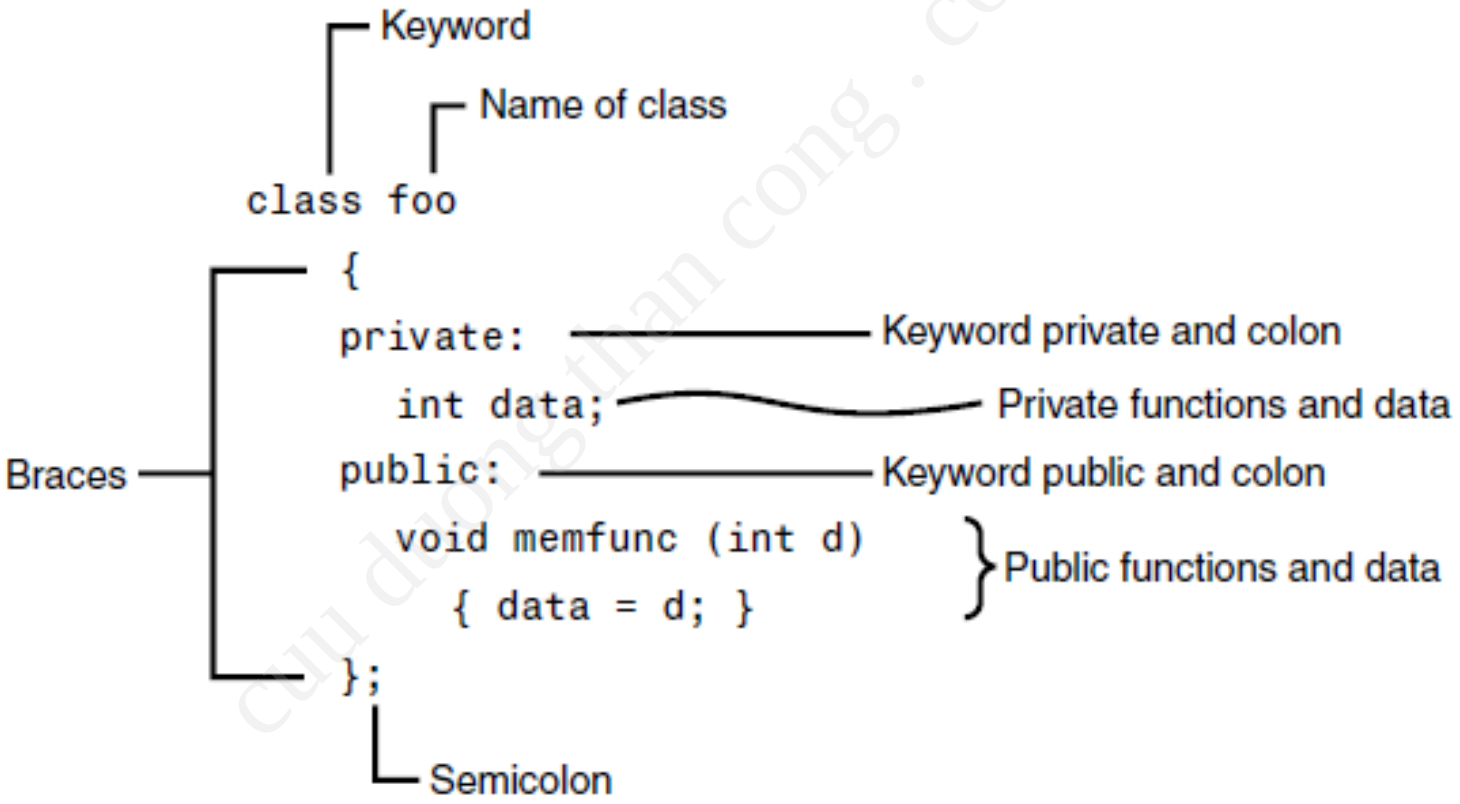
Class Data

- The data items within a class are called data members (or sometimes member data). There can be any number of data members in a class, just as there can be any number of data items in a structure.

Member Function

- Member functions are functions that are included within a class. The function bodies of these functions have been written on the same line as the braces that delimit them.

Functions Are Public, Data Is Private



Using the Class

- The definition of the class does not create any objects. It only describes how they will look when they are created, just as a structure definition describes how a structure will look but doesn't create any structure variables.
- Defining an object is similar to defining a variable of any data type: Space is set aside for it in memory.

```
smallobj s1, s2;
```

Calling Member Functions

- To use a member function, the dot operator connects the object name and the member function. The syntax is similar to the way we refer to structure members, but the parentheses signal that we're executing a member function rather than referring to a data item.

```
s1.setdata(1066);
s2.setdata(1776);
```

C++ Objects as Physical Objects

- In many programming situations, objects in programs represent physical objects: things that can be felt or seen. These situations provide vivid examples of the correspondence between the program and the real world.

```
class part //define class
{
    private:
        int modelnumber; //ID number of widget
        int partnumber; //ID number of widget part
        float cost; //cost of part
    public:
        void setpart(int mn, int pn, float c) //set data
        {
            modelnumber = mn;
            partnumber = pn;
            cost = c;
        }
```

```
    void showpart() //display data
    {
        cout << “Model “ << modelnumber;
        cout << “, part “ << partnumber;
        cout << “, costs $” << cost << endl;
    }
};
```

C++ Objects as Data Types

- C++ objects can represent: variables of a user-defined data type.

Constructors

- Sometimes, however, it's convenient if an object can initialize itself when it's first created, without requiring a separate call to a member function. Automatic initialization is carried out using a special member function called a constructor. A constructor is a member function that is executed automatically whenever an object is created.

Constructors

```
class Counter
{
    private:
        unsigned int count; //count
    public:
        Counter() : count(0) //constructor
        { /*empty body*/ }
        void inc_count() //increment count
        { count++; }
        int get_count() //return count
        { return count; }
};
```

```
int main()
{
    Counter c1, c2; //define and initialize
    cout << “\nc1=” << c1.get_count(); //display
    cout << “\nc2=” << c2.get_count();
    c1.inc_count(); //increment c1
    c2.inc_count(); //increment c2
    c2.inc_count(); //increment c2
    cout << “\nc1=” << c1.get_count(); //display again
    cout << “\nc2=” << c2.get_count();
    cout << endl;
    return 0;
}
```

Destructors

- We've seen that a special member function—the constructor—is called automatically when an object is first created. You might guess that another function is called automatically when an object is destroyed. This is indeed the case. Such a function is called a destructor. A destructor has the same name as the constructor (which is the same as the class name) but is preceded by a tilde.

Destructors

```
class Foo
{
    private:
        int data;
    public:
        Foo() : data(0) //constructor
        { }
        ~Foo()
        { }
};
```

Objects as Function Arguments

- Next, we will demonstrates some new aspects of classes: constructor overloading, defining member functions outside the class, and perhaps most importantly - objects as function arguments.

```
class Distance //English Distance class
{
    private:
        int feet;
        float inches;
    public: //constructor (no args)
        Distance() : feet(0), inches(0.0)
        { }
        Distance(int ft, float in) : feet(ft), inches(in)
        { }
        void getdist() //get length from user
        {
            cout << "\nEnter feet: "; cin >> feet;
            cout << "Enter inches: "; cin >> inches;
        }
```

```
    void showdist() //display distance
    { cout << feet << "\'-" << inches << '\"'; }
    void add_dist( Distance, Distance ); //declaration
};
```

```
void Distance::add_dist(Distance d2, Distance d3)
{
    inches = d2.inches + d3.inches; //add the inches
    feet = 0; //(for possible carry)
    if(inches >= 12.0) //if total exceeds 12.0,
    { //then decrease inches
        inches -= 12.0; //by 12.0 and
        feet++; //increase feet
    } //by 1
    feet += d2.feet + d3.feet; //add the feet
}
```

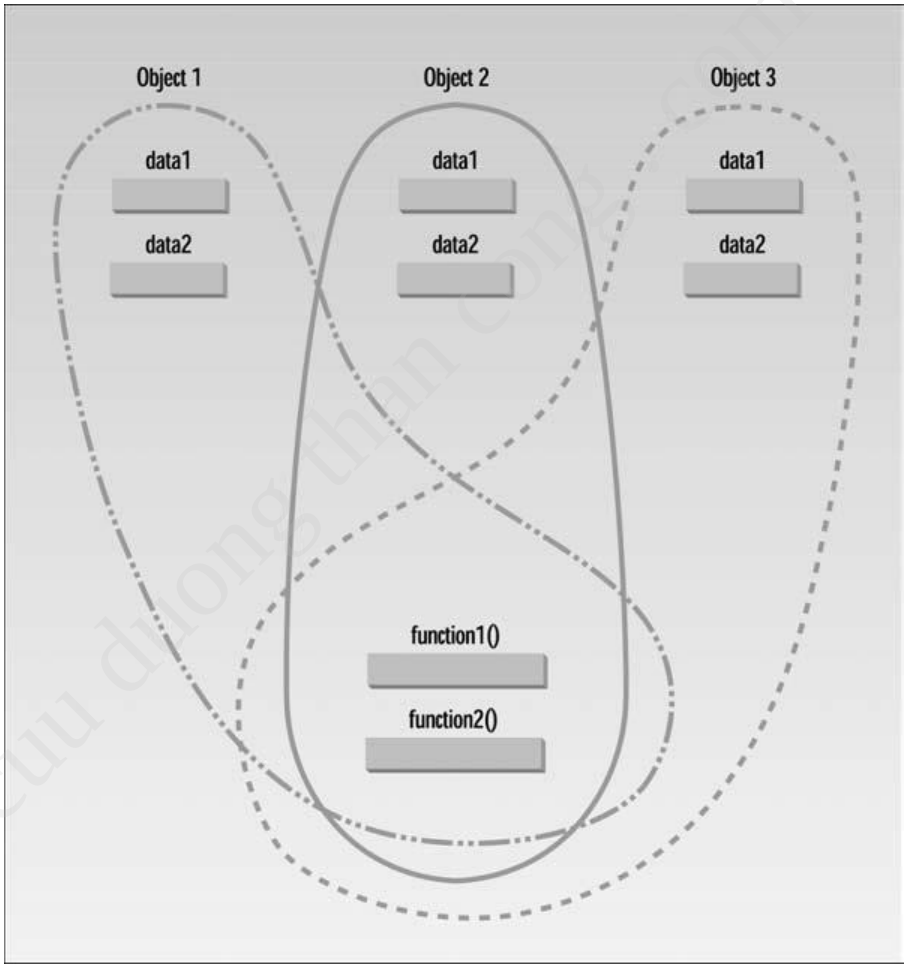
```
int main()
{
    Distance dist1, dist3; //define two lengths
    Distance dist2(11, 6.25); //define and initialize dist2
    dist1.getdist(); //get dist1 from user
    dist3.add_dist(dist1, dist2); //dist3 = dist1 + dist2
    //display all lengths
    cout << “\ndist1 = “; dist1.showdist();
    cout << “\ndist2 = “; dist2.showdist();
    cout << “\ndist3 = “; dist3.showdist();
    cout << endl;
    return 0;
}
```

The Default Copy Constructor

- default copy constructor initialize an object with another object of the same type.

```
Distance dist2(dist1);
```

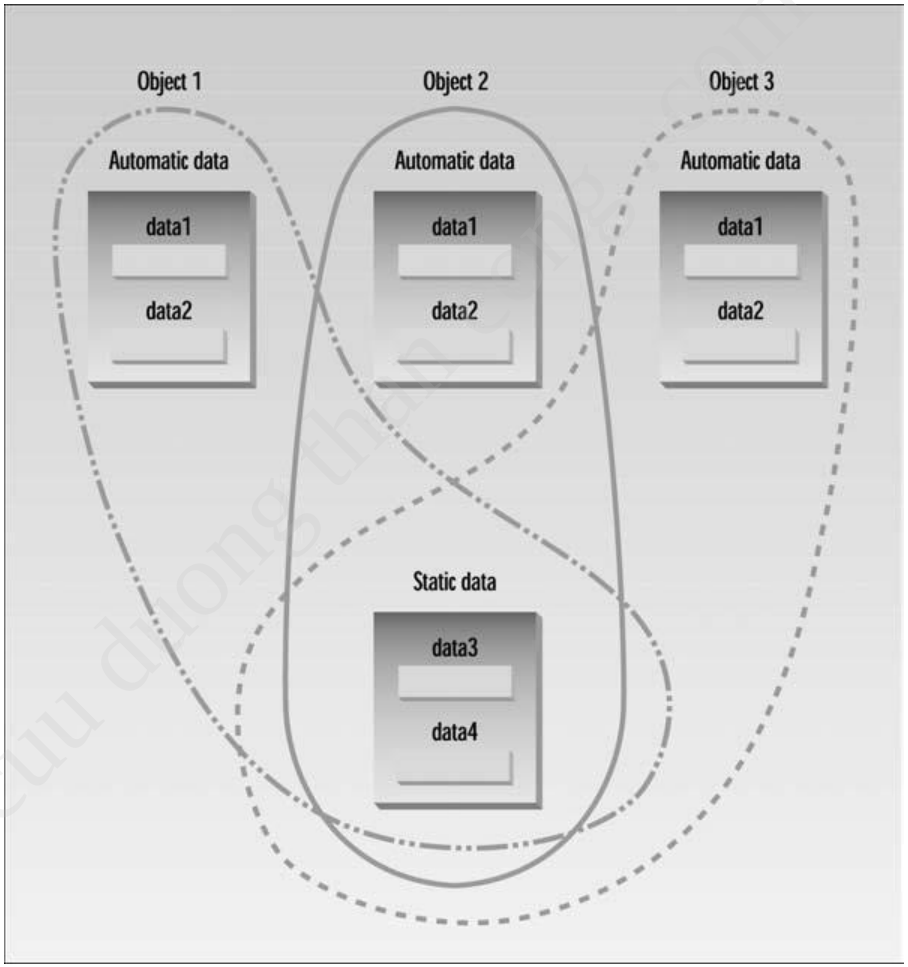
Structures and Classes



Static Class Data

- If a data item in a class is declared as static, only one such item is created for the entire class, no matter how many objects there are. A static data item is useful when all objects of the same class must share a common item of information.

Static Class Data



```
class foo
{
    private:
        static int count; //note: “declaration” only!
    public:
        foo()
        { count++; }
        int getcount() //returns count
        { return count; }
};
int foo::count = 0; //*definition* of count
```

```
int main()
{
    foo f1, f2, f3; //create three objects
    cout << “count is “ << f1.getcount() << endl;
    cout << “count is “ << f2.getcount() << endl;
    cout << “count is “ << f3.getcount() << endl;
    return 0;
}
```

const and Classes

- const Member Functions
- const Member Function Arguments
- const Objects