C++ Objects and classes

Class and object

- A *class* is an expanded concept of a **data structure**: instead of holding only **data**, it can hold both **data** and **functions**.
- An *object* is an instant of a class. In terms of variables, a **class** would be the **type**, and an **object** would be the **variable**

A Simple Class

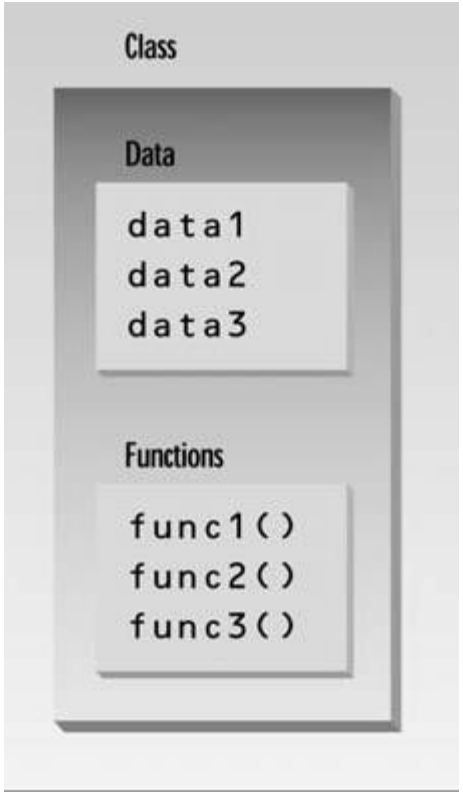
```
// smallobj.cpp
// demonstrates a small, simple object
#include <iostream>
using namespace std;
class smallobj              //define a class
{
 private:
int somedata;               //class data
 public:
 void setdata(int d)    //member function to set data
{ somedata = d; }
void showdata()       //member function to display data
{ cout << “Data is “ << somedata << endl; }
 };
```

```
int main()
{
 smallobj s1, s2;   //define two objects of class smallobj
 s1.setdata(1066);   //call member function to set data
 s2.setdata(1776);
s1.showdata();    //call member function to display data
 s2.showdata();
 return 0;
  }
```

- In **SMALLOBJ**, the **class**—whose name is **smallobj**—is defined in the first part of the program. Later, in main(), we define two **objects**—**s1** and **s2**—that are instances of that class. Here's the output of the
- program:
- Data is 1066 ←□□□ object s1 displayed this
- Data is 1776 ←□□□ object s2 displayed this

Classes contain data and functions.

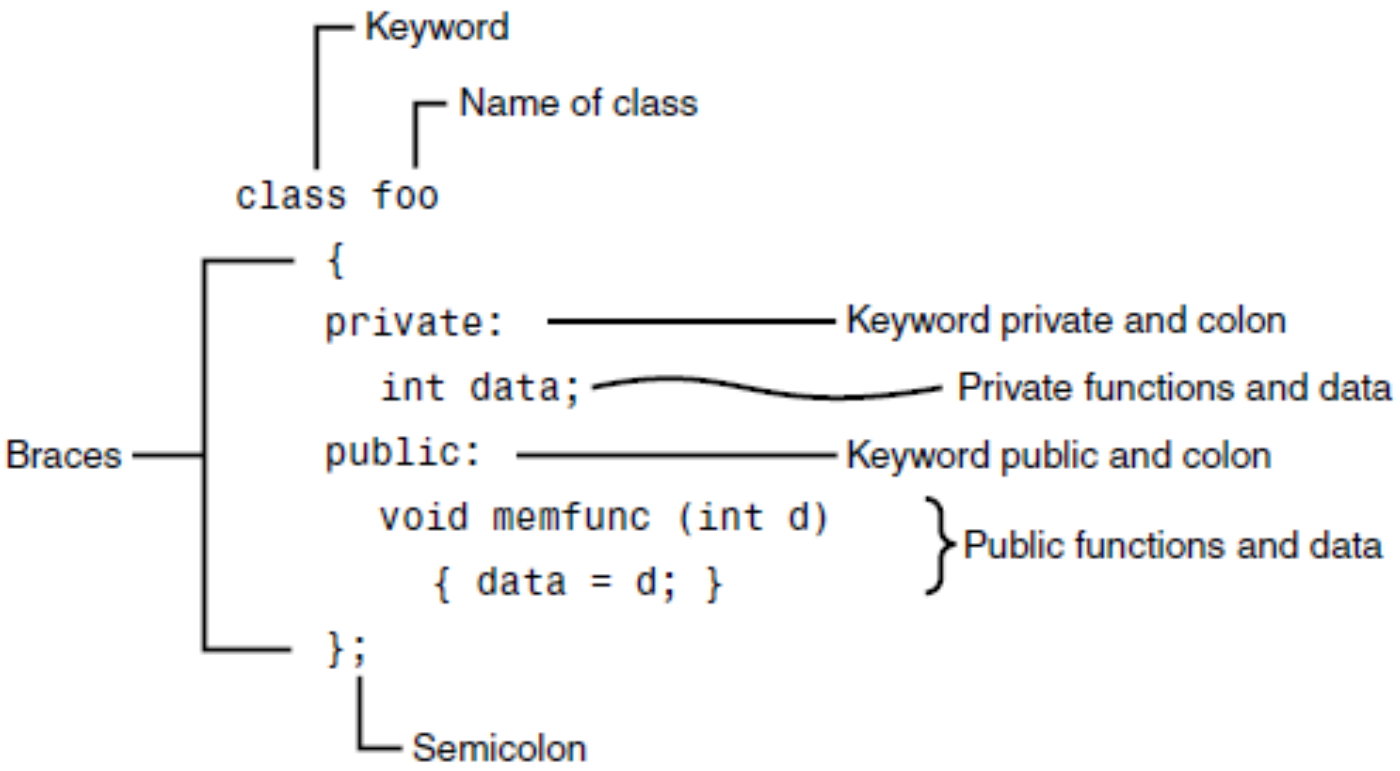
- Placing data and functions **together into single entity** is the **central idea of OOP**.



Defining the Class

```
class smallobj                  //define a class
{
private:
int somedata;                   //class data (data member)
public:
void setdata(int d)             //member function to set data
{ somedata = d; }
void showdata()                 //member function to display data
 { cout << "\nData is " << somedata; }
 };
```

Syntax of a class specifier



Class Data and Member Functions

Class Data:

- The smallobj class contains one data item: somedata, which is of type int. The data items within a class are called *data members.*

Member Functions:

- There are two member functions in smallobj: setdata() and showdata().

```
void setdata(int d)
{
somedata = d;
```

- }

Defining Objects

- The first statement in main()
- smallobj s1, s2;
- defines two objects, s1 and s2, of class smallobj
- Defining an object is similar to defining a variable of any data type: Space is set aside for it in memory. Objects are sometimes called *instance variables*.

Calling Member Functions

- The next two statements in main() call the member function setdata():

  s1.setdata(1066);

  s2.setdata(1776);

- The syntax is used to call a member function that is *associated with a specific object*
- To use a member function, the dot operator (the period) connects the object name and the member function.

Functions Are Public, Data Is Private

- Usually the data within a class is private and the functions are public. This is a result of the way classes are used. The data is hidden so it will be safe from accidental manipulation, while the functions that operate on the data are public so they can be accessed from outside the class.
- However, **there is no rule** that says data must be private and functions public;

example1

- // objpart.cpp
- // widget part as an object
- #include <iostream>
- using namespace std;

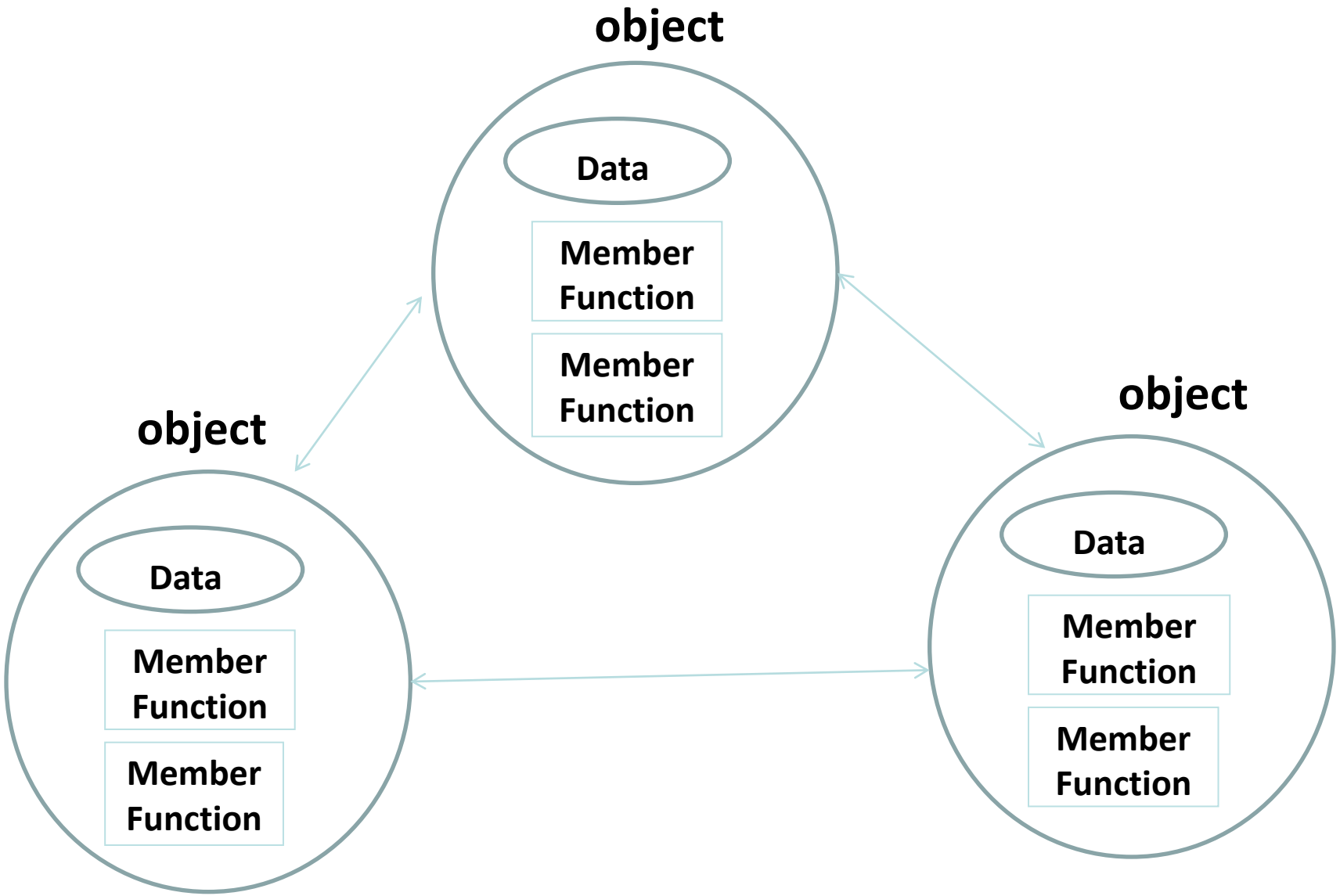
```
class part //define class
{
 private:
 int modelnumber;  //ID number of widget
 int partnumber;  //ID number of widget part
 float cost;              //cost of part
  public:
 void setpart(int mn, int pn, float c) //set data
  {
```

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

- int main()
  {
  Part part1; //define object
  // of class part
  part1.setpart(6244, 373, 217.55F);
  //call member function
  part1.showpart(); //call member function
  return 0;
  }

- In this example only one object of type part is created: part1.
- The member function setpart() sets the three data items in this part to the values 6244, 373, and 217.55.
- The member function showpart() then displays these values. Here’s the output:
- Model 6244, part 373, costs $217.55

THE OBJECT-ORIENTED PARADIGM



CLASSES

- Objects belong to classes
- A class and an object of that class has the same relationship as a data type and a variable
- All objects with the same characteristics (data and functions) constitute one class.
- A class serves only as a plan, or a template, or sketch- of a number of similar things
- .

- It merely specifies what data and what functions will be included in objects of that class
- Declaring a class doesn't create any objects.
- A class is thus a description of a no. of similar objects.
- For instance, HUMAN is a class, and JOHN is its instance (object)

Constructors and destructors

- It is possible to define and at the same time initialize objects of a class.
- A **constructor** is a **member function** that is executed **automatically** whenever an **object** **is created**..
- So an **object** can **initialize itself** when it's first created, without requiring a separate call to a **member function**.

Constructors and destructors

- This **constructor** **function** must have the **same name** as the **class**, and **cannot** have any **return type**;

```
// counter.cpp
// object represents a counter variable
 #include <iostream>
 using namespace std;
 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 0
  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 1
cout << "\nc2=" << c2.get_count();

cout << endl;

return 0;
}
```

- The **Counter** class has **one** data member: count, of type unsigned int (since the count is always positive). It has **three** member functions:
  - the **constructor Counter**(),
  - **inc_count()**, which adds 1 to count;
  - and **get_count()**, which returns the current value of count.

- Thus in main() the statement
- Counter c1, c2;
- creates two objects of type Counter. As each is created, its constructor, Counter(), is executed.This function sets the count variable to 0. So the effect of this single statement is to not only create two objects, but also to initialize their count variables to 0.

Same Name as the Class

- constructor functions.
- **First**, it is no accident that they have **exactly the same name** (Counter in this example) as the class of which they are members
- **Second**, **no return** type is used for constructors

- Here's how you should initialize a data member:

  counter() : count(0)

  { }

- we can rewrite the constructor to print a message when it executes.

  Counter() : count(0)

  { cout << "I'm the constructor\n"; }

Initializing multiple members

- If multiple members must be initialized, they're separated by commas. The result is the *initializer list* (sometimes called by other names, such as the *member-initialization list*).

  Some Class() : m1(7), m2(33), m2(4)

  { }

Counter Output

- the **Counter** class creates **two** counters, **c1** and **c2**.
- It causes the counters to display their initial values, which—as arranged by the constructor—are **0**. It then increments **c1** **once** and **c2** **twice**, and again causes the counters to **display** themselves

Destructors

- Just as a **constructor** is used to **initialize** an **object** when it is created, a **destructor** is used to **clean up** the **object** just before it is destroyed.

- A **destructor** always has the **same name** as the **class** itself, but is **preceded** with a **~** symbol. Unlike constructors,

- a **class** may have at most **one destructor**.
- A **destructor never** takes any **arguments** and has **no explicit return type.**

Destructors

- class Foo
- {
- private:
- int data;
- public:
- Foo() : data(0) //**constructor** (same name as class)
- { }

- **~Foo()** //destructor (same name with tilde)
- { }
- **Like constructors**, **destructors** **do not have a return value**. They also take **no arguments** (the assumption being that there's **only one way to destroy an object**).};