9.2 `Time` Class Case Study (cont.)

Object Size

- People new to object-oriented programming often suppose that objects must be quite large because they contain data members and member functions.
- *Logically*, this is true—you may think of objects as containing data and functions (and our discussion has certainly encouraged this view); *physically*, however, this is not true.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Performance Tip 9.2

Objects contain only data, so objects are much smaller than if they also contained member functions. The compiler creates one copy (only) of the member functions separate from all objects of the class. All objects of the class share this one copy. Each object, of course, needs its own copy of the class's data, because the data can vary among the objects. The function code is nonmodifiable and, hence, can be shared among all objects of one class.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.3 Class Scope and Accessing Class Members

- A class's data members and member functions belong to that class's scope.
- Nonmember functions are defined at *global namespace scope*, by default.
- Within a class's scope, class members are immediately accessible by all of that class's member functions and can be referenced by name.
- Outside a class's scope, `public` class members are referenced through one of the handles on an object—an *object name*, a *reference* to an object or a *pointer* to an object.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.3 Class Scope and Accessing Class Members (cont.)

Class Scope and Block Scope

- If a member function defines a variable with the same name as a variable with class scope, the class-scope variable is *hidden* in the function by the block-scope variable.
  - Such a hidden variable can be accessed by preceding the variable name with the class name followed by the scope resolution operator (`::`).

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.3 Class Scope and Accessing Class Members (cont.)

Dot (`.`) and Arrow (`->`) Member Selection Operators

- The dot member selection operator (`.`) is preceded by an object's name or with a reference to an object to access the object's members.
- The arrow member selection operator (`->`) is preceded by a pointer to an object to access the object's members.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.3 Class Scope and Accessing Class Members (cont.)

Accessing public Class Members Through Objects, References and Pointers

- Consider an Account class that has a public `setBalance` member function. Given the following declarations:

```
Account account; // an Account object
// accountRef refers to an Account object
Account &accountRef = account;
// accountPtr points to an Account object
Account *accountPtr = &account;
```

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.3 Class Scope and Accessing Class Members (cont.)

You can invoke member function setBalance using the dot (.) and arrow (->) member selection operators as follows:

```
// call setBalance via the Account object
account.setBalance( 123.45 );
// call setBalance via a reference to the
Account object
accountRef.setBalance( 123.45 );
// call setBalance via a pointer to the Account
object
accountPtr->setBalance( 123.45 );
```

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.4 Access Functions and Utility Functions

Access Functions

- Access functions can read or display data.
- A common use for access functions is to test the truth or falsity of conditions—such functions are often called predicate functions.

Utility Functions

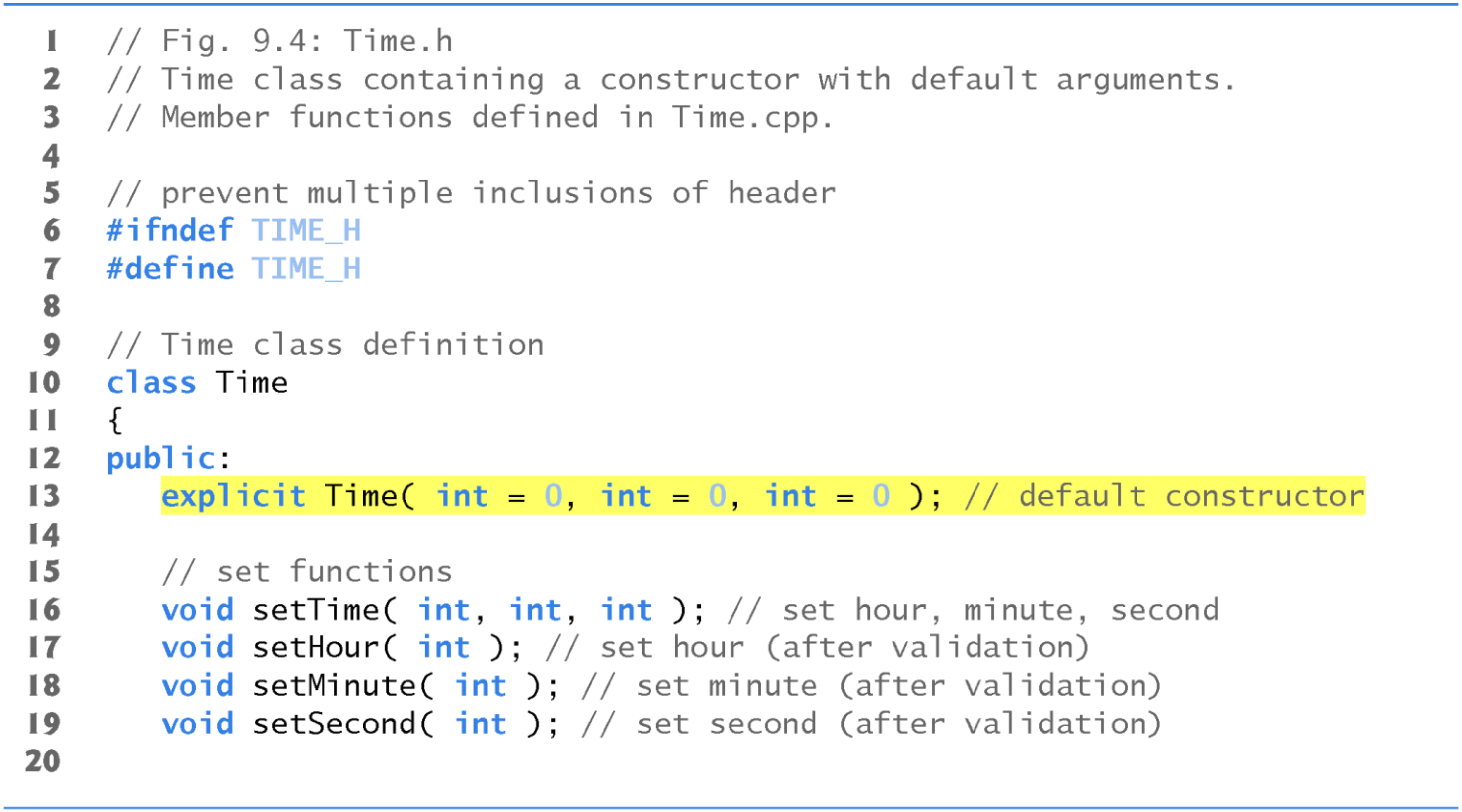
- A utility function (also called a helper function) is a `private` member function that supports the operation of the class's other member functions.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.5 `Time` Class Case Study: Constructors with Default Arguments

- The program of Figs. 9.4–9.6 enhances class `Time` to demonstrate how arguments are implicitly passed to a constructor.
- The constructor defined in Fig. 9.2 initialized `hour`, `minute` and `second` to 0 (i.e., midnight in universal time).
- Like other functions, constructors can specify *default arguments.*

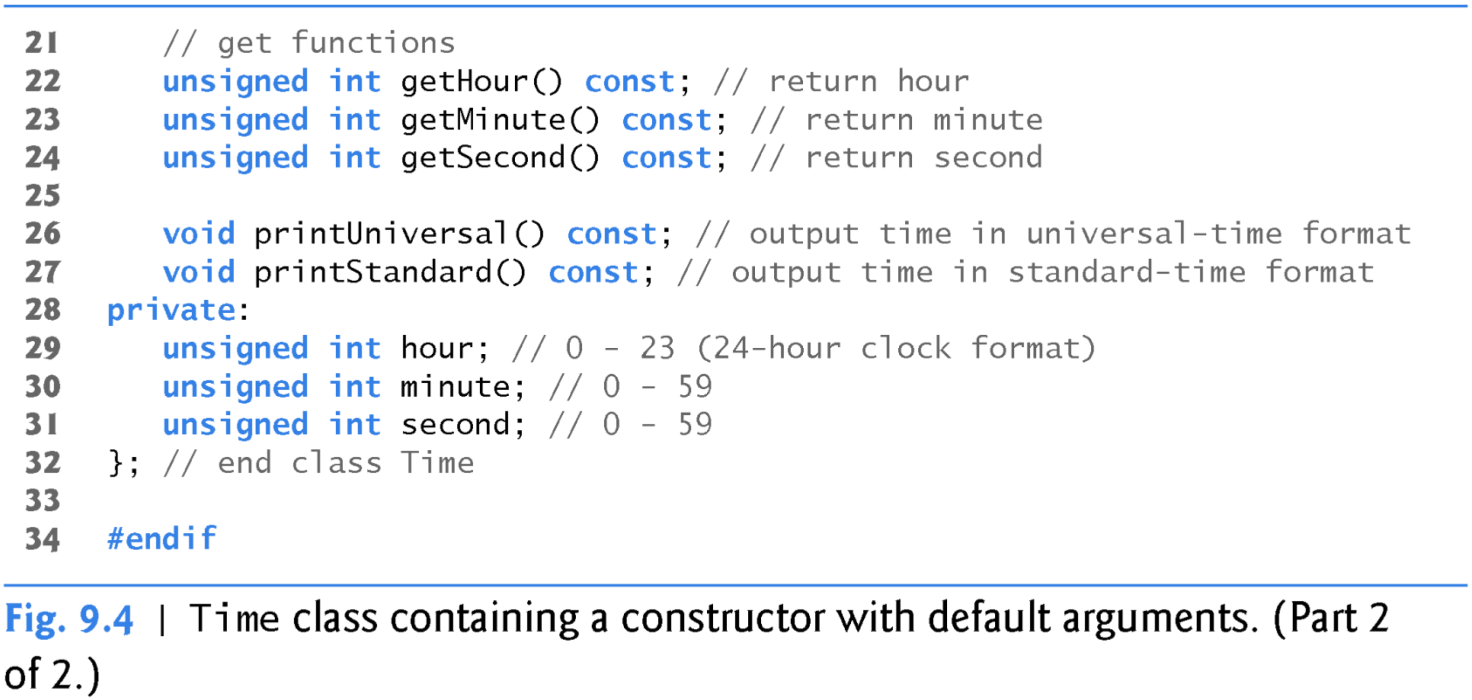
©1992-2014 by Pearson Education, Inc. All Rights Reserved.



```
// Fig. 9.4: Time.h
// Time class containing a constructor with default arguments.
// Member functions defined in Time.cpp.

// prevent multiple inclusions of header
#ifndef TIME_H
#define TIME_H

// Time class definition
class Time
{
public:
   explicit Time( int = 0, int = 0, int = 0 ); // default constructor

   // set functions
   void setTime( int, int, int ); // set hour, minute, second
   void setHour( int ); // set hour (after validation)
   void setMinute( int ); // set minute (after validation)
   void setSecond( int ); // set second (after validation)

```

Fig. 9.4 | Time class containing a constructor with default arguments. (Part 1 of 2.)

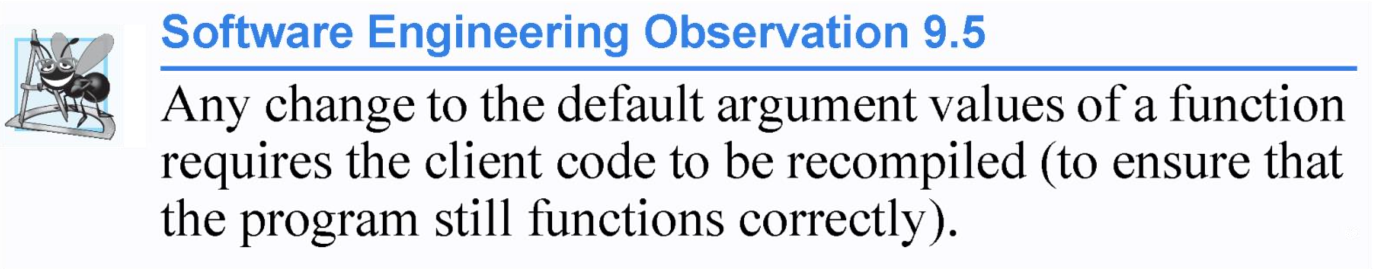
©1992-2014 by Pearson Education, Inc. All
Rights Reserved.



```cpp
   // get functions
   unsigned int getHour() const; // return hour
   unsigned int getMinute() const; // return minute
   unsigned int getSecond() const; // return second

   void printUniversal() const; // output time in universal-time format
   void printStandard() const; // output time in standard-time format
private:
   unsigned int hour; // 0 - 23 (24-hour clock format)
   unsigned int minute; // 0 - 59
   unsigned int second; // 0 - 59
}; // end class Time

#endif
```

Fig. 9.4 | Time class containing a constructor with default arguments. (Part 2 of 2.)

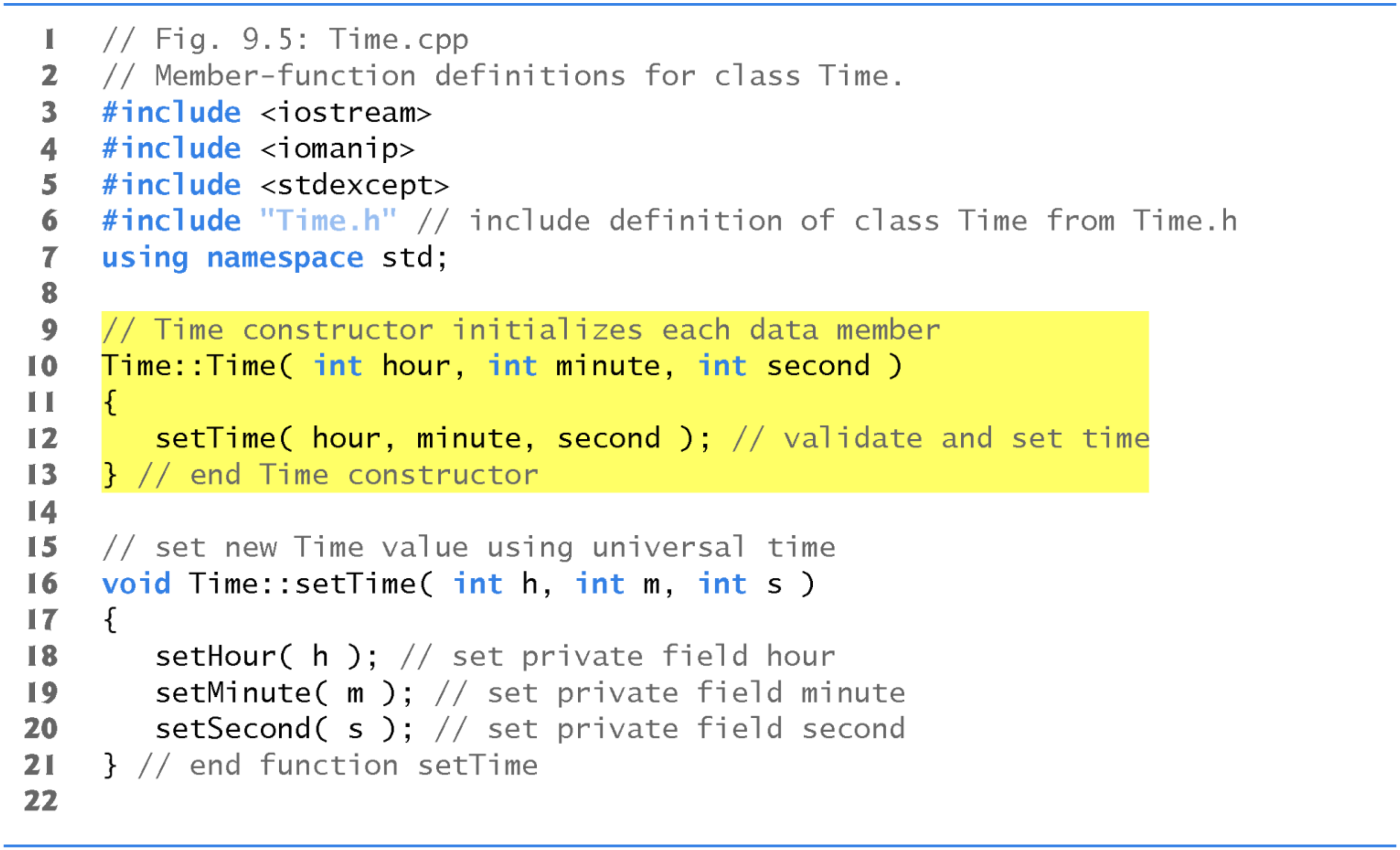
©1992-2014 by Pearson Education, Inc. All
Rights Reserved.



Software Engineering Observation 9.5

Any change to the default argument values of a function requires the client code to be recompiled (to ensure that the program still functions correctly).

©1992-2014 by Pearson Education, Inc. All Rights Reserved.



```cpp
// Fig. 9.5: Time.cpp
// Member-function definitions for class Time.
#include <iostream>
#include <iomanip>
#include <stdexcept>
#include "Time.h" // include definition of class Time from Time.h
using namespace std;

// Time constructor initializes each data member
Time::Time( int hour, int minute, int second )
{
   setTime( hour, minute, second ); // validate and set time
} // end Time constructor

// set new Time value using universal time
void Time::setTime( int h, int m, int s )
{
   setHour( h ); // set private field hour
   setMinute( m ); // set private field minute
   setSecond( s ); // set private field second
} // end function setTime

```

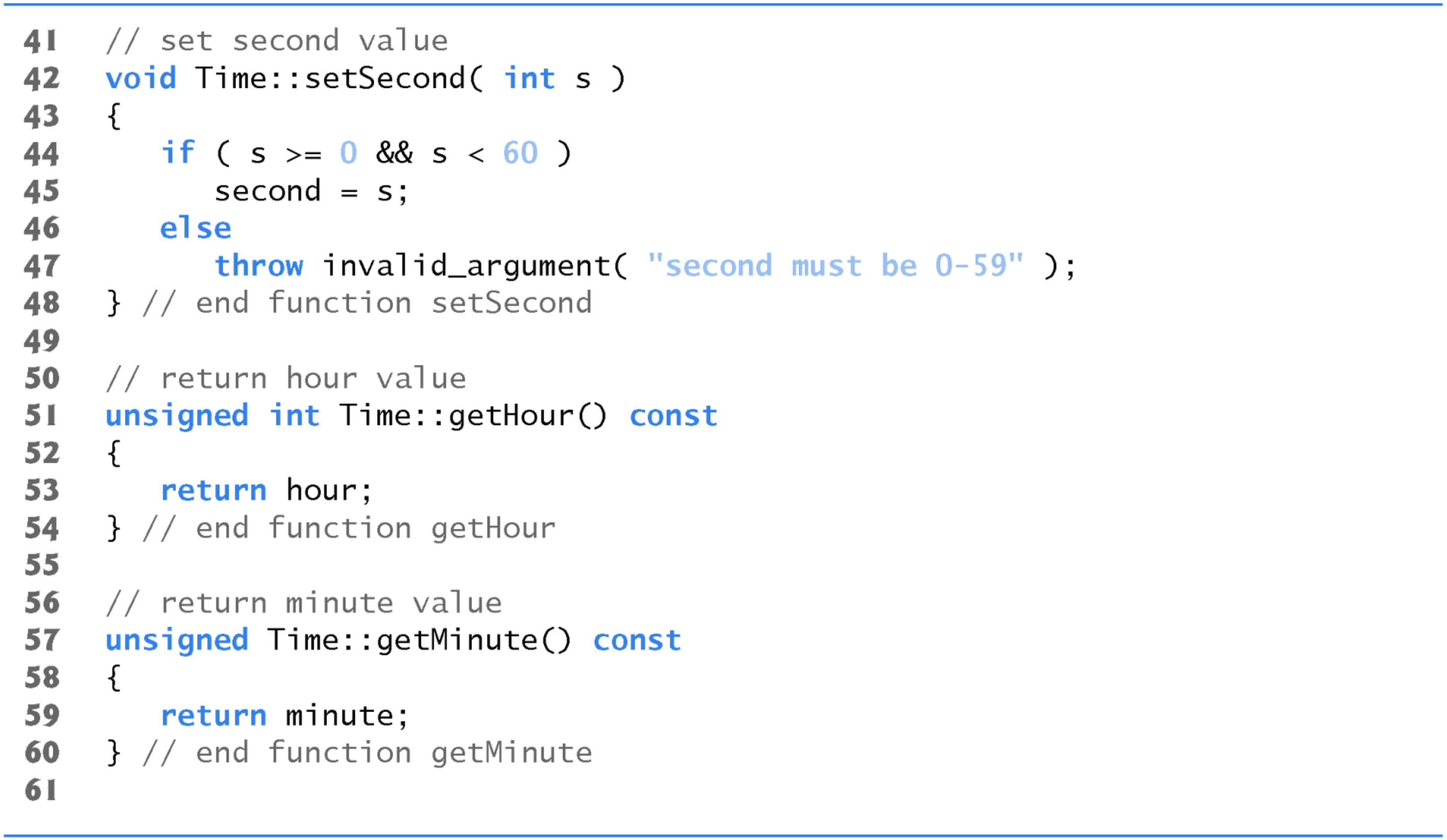
Fig. 9.5 | Member-function definitions for class Time. (Part 1 of 4.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
// set hour value
void Time::setHour( int h )
{
   if ( h >= 0 && h < 24 )
      hour = h;
   else
      throw invalid_argument( "hour must be 0-23" );
} // end function setHour

// set minute value
void Time::setMinute( int m )
{
   if ( m >= 0 && m < 60 )
      minute = m;
   else
      throw invalid_argument( "minute must be 0-59" );
} // end function setMinute

```

Fig. 9.5 | Member-function definitions for class `Time`. (Part 2 of 4.)

©1992-2014 by Pearson Education, Inc. All
Rights Reserved.



```cpp
// set second value
void Time::setSecond( int s )
{
   if ( s >= 0 && s < 60 )
      second = s;
   else
      throw invalid_argument( "second must be 0-59" );
} // end function setSecond

// return hour value
unsigned int Time::getHour() const
{
   return hour;
} // end function getHour

// return minute value
unsigned Time::getMinute() const
{
   return minute;
} // end function getMinute

```

Fig. 9.5 | Member-function definitions for class `Time`. (Part 3 of 4.)

©1992-2014 by Pearson Education, Inc. All
Rights Reserved.