```
23      cout << "\n\n*********************************************\n"
24         << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
25         << "t.badSetHour( 12 ) as an lvalue, invalid hour: "
26         << t.getHour()
27         << "\n*********************************************" << endl;
28   } // end main
```

```
Valid hour before modification: 20
Invalid hour after modification: 30

*********************************************
POOR PROGRAMMING PRACTICE!!!!!!!!
t.badSetHour( 12 ) as an lvalue, invalid hour: 74
*********************************************
```

Fig. 9.12 | `public` member function that returns a reference to a `private` data member. (Part 2 of 2.)

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

9.9 Default Memberwise Assignment

- The assignment operator (=) can be used to assign an object to another object of the same type.
- By default, such assignment is performed by memberwise assignment (also called copy assignment).
  - Each data member of the object on the right of the assignment operator is assigned individually to the *same* data member in the object on the *left* of the assignment operator.
- [*Caution:* Memberwise assignment can cause serious problems when used with a class whose data members contain pointers to dynamically allocated memory; we discuss these problems in Chapter 10 and show how to deal with them.]

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

```
1  // Fig. 9.13: Date.h
2  // Date class declaration.  Member functions are defined in Date.cpp.
3
4  // prevent multiple inclusions of header
5  #ifndef DATE_H
6  #define DATE_H
7
8  // class Date definition
9  class Date
10 {
11 public:
12    explicit Date( int = 1, int = 1, int = 2000 ); // default constructor
13    void print();
14 private:
15    unsigned int month;
16    unsigned int day;
17    unsigned int year;
18 }; // end class Date
19
20 #endif
```

Fig. 9.13 | `Date` class declaration.

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

```
1  // Fig. 9.14: Date.cpp
2  // Date class member-function definitions.
3  #include <iostream>
4  #include "Date.h" // include definition of class Date from Date.h
5  using namespace std;
6
7  // Date constructor (should do range checking)
8  Date::Date( int m, int d, int y )
9     : month( m ), day( d ), year( y )
10 {
11 } // end constructor Date
12
13 // print Date in the format mm/dd/yyyy
14 void Date::print()
15 {
16    cout << month << '/' << day << '/' << year;
17 } // end function print
```

Fig. 9.14 | `Date` class member-function definitions.

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

```
1  // Fig. 9.15: fig09_15.cpp
2  // Demonstrating that class objects can be assigned
3  // to each other using default memberwise assignment.
4  #include <iostream>
5  #include "Date.h" // include definition of class Date from Date.h
6  using namespace std;
7
8  int main()
9  {
10    Date date1( 7, 4, 2004 );
11    Date date2; // date2 defaults to 1/1/2000
12
13    cout << "date1 = ";
14    date1.print();
15    cout << "\ndate2 = ";
16    date2.print();
17
18    date2 = date1; // default memberwise assignment
19
20    cout << "\n\nAfter default memberwise assignment, date2 = ";
21    date2.print();
22    cout << endl;
23 } // end main
```

Fig. 9.15 | Class objects can be assigned to each other using default memberwise assignment. (Part 1 of 2.)

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

```
date1 = 7/4/2004
date2 = 1/1/2000

After default memberwise assignment, date2 = 7/4/2004
```

Fig. 9.15 | Class objects can be assigned to each other using default memberwise assignment. (Part 2 of 2.)

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

9.9 Default Memberwise Assignment (cont.)

- Objects may be passed as function arguments and may be returned from functions.
- Such passing and returning is performed using pass-by-value by default—a *copy* of the object is passed or returned.
  - C++ creates a new object and uses a copy constructor to copy the original object's values into the new object.
- For each class, the compiler provides a default copy constructor that copies each member of the original object into the corresponding member of the new object.
  - Copy constructors can cause serious problems when used with a class whose data members contain pointers to dynamically allocated memory.
- Chapter 10 discusses customized copy constructors.

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

9.10 `const` Objects and `const` Member Functions

- Some objects need to be modifiable and some do not.
- You may use keyword `const` to specify that an object *is not* modifiable and that any attempt to modify the object should result in a compilation error.
- The statement

```
const Time noon( 12, 0, 0 );
```

declares a `const` object `noon` of class `Time` and initializes it to 12 noon. It's possible to instantiate `const` and non-`const` objects of the same class.

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

Software Engineering Observation 9.8

Attempts to modify a `const` object are caught at compile time rather than causing execution-time errors.

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

Performance Tip 9.3

Declaring variables and objects `const` when appropriate can improve performance—compilers can perform optimizations on constants that cannot be performed on non-`const` variables.

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

9.10 `const` Objects and `const` Member Functions (cont.)

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

9.10 `const` Objects and `const` Member Functions (cont.)

- C++ *disallows member function calls for* `const` *objects unless the member functions themselves are also declared* `const`.
- This is true even for *get* member functions that do *not* modify the object.
- *This is also a key reason that we've declared as* `const` *all member-functions that do not modify the objects on which they're called.*
- A member function is specified as `const` both in its prototype by inserting the keyword `const` *after* the function's parameter list and, in the case of the function definition, before the left brace that begins the function *body*.

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

Common Programming Error 9.2

Defining as `const` a member function that modifies a data member of the object is a compilation error.

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

Common Programming Error 9.3

Defining as `const` a member function that calls a non-`const` member function of the class on the same object is a compilation error.

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

Common Programming Error 9.4

Invoking a non-`const` member function on a `const` object is a compilation error.

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