Computer Programming
Chapter 9: Class

Hung-Yu Wei
Department of Electrical Engineering
National Taiwan University

[review] Constructor

- Constructor
  - A special member function
  - Used to initialize an object
  - We can use constructor to initialize data members
- Same name as the class

```
Time::Time()
{

}
```

- No return type
  - Not even *void*

[review]Example: constructor

```
class Time
{
  public:

    Time( )  //beginning of the constructor
    {
        Initialization when a Time object is created

    } // end of the constructor

  other part of the class.....
}
```

9.7 Access Functions and Utility Functions

- Access functions
  - Declared *public*
  - Use to
    - Read data
    - Display data
    - Test the truth or falsity of conditions
- Utility functions
  - Usually declared as *private*
  - Also know as *helper function*
  - To support (help) public member function

```
1  // Fig. 9.7: SalesPerson.h
2  // SalesPerson class definition.
3  // Member functions defined in SalesPerson.cpp.
4  #ifndef SALESP_H
5  #define SALESP_H
6
7  class SalesPerson
8  {
9  public:
10    static const int monthsPerYear = 12; // months in one year
11    SalesPerson(); // constructor
12    void getSalesFromUser(); // input sales from keyboard
13    void setSales( int, double ); // set sales for a specific month
14    void printAnnualSales(); // summarize and print sales
15 private:
16    double totalAnnualSales(); // prototype for utility function
17    double sales[ monthsPerYear ]; // 12 monthly sales figures
18 }; // end class SalesPerson
19
20 #endif
```

Fig. 9.7 | SalesPerson class definition.

```
1  // Fig. 9.8: SalesPerson.cpp
2  // SalesPerson class member-function definitions.
3  #include <iostream>
4  #include <iomanip>
5  #include "SalesPerson.h" // include SalesPerson class definition
6  using namespace std;
7
8  // initialize elements of array sales to 0.0
9  SalesPerson::SalesPerson()
10 {
11    for ( int i = 0; i < monthsPerYear; i++ )
12       sales[ i ] = 0.0;
13 } // end SalesPerson constructor
14
```

Fig. 9.8 | SalesPerson class member-function definitions. (Part 1 of 3.)

```
15  // get 12 sales figures from the user at the keyboard
16  void SalesPerson::getSalesFromUser()
17  {
18     double salesFigure;
19
20     for ( int i = 1; i <= monthsPerYear; i++ )
21     {
22        cout << "Enter sales amount for month " << i << ": ";
23        cin >> salesFigure;
24        setSales( i, salesFigure );
25     } // end for
26  } // end function getSalesFromUser
27
28  // set one of the 12 monthly sales figures; function subtracts
29  // one from month value for proper subscript in sales array
30  void SalesPerson::setSales( int month, double amount )
31  {
32     // test for valid month and amount values
33     if ( month >= 1 && month <= monthsPerYear && amount > 0 )
34        sales[ month - 1 ] = amount; // adjust for subscripts 0-11
35     else // invalid month or amount value
36        cout << "Invalid month or sales figure" << endl;
37  } // end function setSales
```


Fig. 9.8 | `SalesPerson` class member-function definitions. (Part 2 of 3.)

```
38
39  // print total annual sales (with the help of utility function)
40  void SalesPerson::printAnnualSales()
41  {
42     cout << setprecision( 2 ) << fixed
43        << "\nThe total annual sales are: $"
44        << totalAnnualSales() << endl; // call utility function
45  } // end function printAnnualSales
46
47  // private utility function to total annual sales
48  double SalesPerson::totalAnnualSales()
49  {
50     double total = 0.0; // initialize total
51
52     for ( int i = 0; i < monthsPerYear; i++ ) // summarize sales results
53        total += sales[ i ]; // add month i sales to total
54
55     return total;
56  } // end function totalAnnualSales
```

Fig. 9.8 | `SalesPerson` class member-function definitions. (Part 3 of 3.)

```
1  // Fig. 9.9: fig09_09.cpp
2  // Utility function demonstration.
3  // Compile this program with SalesPerson.cpp
4
5  // include SalesPerson class definition from SalesPerson.h
6  #include "SalesPerson.h"
7
8  int main()
9  {
10    SalesPerson s; // create SalesPerson object s
11
12    s.getSalesFromUser(); // note simple sequential code; there a
13    s.printAnnualSales(); // no control statements in main
14 } // end main
```

Fig. 9.9 | Utility function demonstration. (Part 1 of 2.)

9.8 constructor with default arguments

- Constructor with **Default Arguments**
  - Constructor is called automatically when an object is created (declared)
  - Use for initialization
    - Default argument is useful for initialization
- Access functions
  - Set
    - For example (setHour, setMinute, setSecond)
  - Get
- Validity check with "set" member functions

```
1  // Fig. 9.10: Time.h
2  // Time class containing a constructor with default arguments.
3  // Member functions defined in Time.cpp.
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Time abstract data type definition
10 class Time
11 {
12 public:
13    Time( int = 0, int = 0, int = 0 ); // default constructor
14    void setTime( int, int, int ); // set hour, minute, second
15    void printUniversal(); // output time in universal-time format
16    void printStandard(); // output time in standard-time format
17 private:
18    int hour; // 0 - 23 (24-hour clock format)
19    int minute; // 0 - 59
20    int second; // 0 - 59
21 }; // end class Time
22
23 #endif
```

Fig. 9.10 | `Time` class containing a constructor with default arguments.

```
1  // Fig.9.11: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  #include <iomanip>
5  #include "Time.h" // include definition of class Time from Time.h
6  using namespace std;
7
8  // Time constructor initializes each data member to zero;
9  // ensures that Time objects start in a consistent state
10 Time::Time( int hr, int min, int sec )
11 {
12    setTime( hr, min, sec ); // validate and set time
13 } // end Time constructor
14
15 // set new Time value using universal time; ensure that
16 // the data remains consistent by setting invalid values to zero
17 void Time::setTime( int h, int m, int s )
18 {
19    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
20    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
21    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
22 } // end function setTime
```

Fig. 9.11 | `Time` class member-function definitions including a constructor that takes arguments. (Part 1 of 2.)

```
23
24  // print Time in universal-time format (HH:MM:SS)
25  void Time::printUniversal()
26  {
27     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
28        << setw( 2 ) << minute << ":" << setw( 2 ) << second;
29  } // end function printUniversal
30
31  // print Time in standard-time format (HH:MM:SS AM or PM)
32  void Time::printStandard()
33  {
34     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
35        << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
36        << second << ( hour < 12 ? " AM" : " PM" );
37  } // end function printStandard
```

Fig. 9.11 | `Time` class member-function definitions including a constructor that takes arguments. (Part 2 of 2.)

```
1  // Fig. 9.12: fig09_12.cpp
2  // Demonstrating a default constructor for class Time.
3  #include <iostream>
4  #include "Time.h" // include definition of class Time from Time.h
5  using namespace std;
6
7  int main()
8  {
9     Time t1; // all arguments defaulted
10    Time t2( 2 ); // hour specified; minute and second defaulted
11    Time t3( 21, 34 ); // hour and minute specified; second defaulted
12    Time t4( 12, 25, 42 ); // hour, minute and second specified
13    Time t5( 27, 74, 99 ); // all bad values specified
14
15    cout << "Constructed with:\n\nt1: all arguments defaulted\n  ";
16    t1.printUniversal(); // 00:00:00
17    cout << "\n  ";
18    t1.printStandard(); // 12:00:00 AM
19
20    cout << "\n\nt2: hour specified; minute and second defaulted\n  ";
21    t2.printUniversal(); // 02:00:00
22    cout << "\n  ";
23    t2.printStandard(); // 2:00:00 AM
```

Fig. 9.12 | Constructor with default arguments. (Part 1 of 3.)

```
24
25      cout << "\n\nt3: hour and minute specified; second defaulted\n  ";
26      t3.printUniversal(); // 21:34:00
27      cout << "\n  ";
28      t3.printStandard(); // 9:34:00 PM
29
30      cout << "\n\nt4: hour, minute and second specified\n  ";
31      t4.printUniversal(); // 12:25:42
32      cout << "\n  ";
33      t4.printStandard(); // 12:25:42 PM
34
35      cout << "\n\nt5: all invalid values specified\n  ";
36      t5.printUniversal(); // 00:00:00
37      cout << "\n  ";
38      t5.printStandard(); // 12:00:00 AM
39      cout << endl;
40  } // end main
```

Fig. 9.12 | Constructor with default arguments. (Part 2 of 3.)

9.9 Destructor

- Similar to Constructor
  - A special member function
  - Constructor : called when an object is created
  - Destructor : called when an object is destroyed
    - Used for termination housekeeping
- Syntax

  ~ ClassName ()
- Receives no parameters and returns no value.
  - No return type
    - not even void.
- Only 1 destructor per class
- Must be public.

9.10 When Constructors and Destructors Are Called

- Exit
  - Force a program to terminate immediately
  - Does not execute the destructors of automatic objects
- Abort
  - Similar to exit
  - Does not execute any destructor

Static, local, global

- Review: variables
  - Local variable
  - Global variable
  - **static** --- will not be destroyed

- Objects
  - Static objects
  - Local objects (e.g. within a function)
  - Global objects

```
1  // Fig. 9.13: CreateAndDestroy.h
2  // CreateAndDestroy class definition.
3  // Member functions defined in CreateAndDestroy.cpp.
4  #include <string>
5  using namespace std;
6
7  #ifndef CREATE_H
8  #define CREATE_H
9
10 class CreateAndDestroy
11 {
12 public:
13    CreateAndDestroy( int, string ); // constructor
14    ~CreateAndDestroy(); // destructor
15 private:
16    int objectID; // ID number for object
17    string message; // message describing object
18 }; // end class CreateAndDestroy
19
20 #endif
```

Fig. 9.13 | CreateAndDestroy class definition.

```
1  // Fig. 9.14: CreateAndDestroy.cpp
2  // CreateAndDestroy class member-function definitions.
3  #include <iostream>
4  #include "CreateAndDestroy.h"// include CreateAndDestroy class definition
5  using namespace std;
6
7  // constructor
8  CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
9  {
10    objectID = ID; // set object's ID number
11    message = messageString; // set object's descriptive message
12
13    cout << "Object " << objectID << "   constructor runs   "
14       << message << endl;
15 } // end CreateAndDestroy constructor
16
17 // destructor
18 CreateAndDestroy::~CreateAndDestroy()
19 {
20    // output newline for certain objects; helps readability
21    cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
22
23    cout << "Object " << objectID << "   destructor runs    "
24       << message << endl;
25 } // end ~CreateAndDestroy destructor
```


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

```
1  // Fig. 9.15: fig09_15.cpp
2  // Demonstrating the order in which constructors and
3  // destructors are called.
4  #include <iostream>
5  #include "CreateAndDestroy.h" // include CreateAndDestroy class definition
6  using namespace std;
7
8  void create( void ); // prototype
9
10 CreateAndDestroy first( 1, "(global before main)" ); // global object
11
12 int main()
13 {
14    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
15    CreateAndDestroy second( 2, "(local automatic in main)" );
16    static CreateAndDestroy third( 3, "(local static in main)" );
17
18    create(); // call function to create objects
19
20    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
21    CreateAndDestroy fourth( 4, "(local automatic in main)" );
22    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
23 } // end main
```

Fig. 9.15 | Order in which constructors and destructors are called. (Part 1 of 3.)

```
24
25  // function to create objects
26  void create( void )
27  {
28     cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
29     CreateAndDestroy fifth( 5, "(local automatic in create)" );
30     static CreateAndDestroy sixth( 6, "(local static in create)" );
31     CreateAndDestroy seventh( 7, "(local automatic in create)" );
32     cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
33  } // end function create
```

Fig. 9.15 | Order in which constructors and destructors are called. (Part 2 of 3.)

```
Object 1   constructor runs   (global before main)

MAIN FUNCTION: EXECUTION BEGINS
Object 2   constructor runs   (local automatic in main)
Object 3   constructor runs   (local static in main)

CREATE FUNCTION: EXECUTION BEGINS
Object 5   constructor runs   (local automatic in create)
Object 6   constructor runs   (local static in create)
Object 7   constructor runs   (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS
Object 7   destructor runs    (local automatic in create)
Object 5   destructor runs    (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES
Object 4   constructor runs   (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS
Object 4   destructor runs    (local automatic in main)
Object 2   destructor runs    (local automatic in main)

Object 6   destructor runs    (local static in create)
Object 3   destructor runs    (local static in main)

Object 1   destructor runs    (global before main)
```


Fig. 9.15 | Order in which constructors and destructors are called. (Part 3 of 3.)

9.11 Set and Get Function

- Set function
  - Set the values of data member
- Get function
  - Get the value of data member

- A useful interface
  - Work well with private data member
  - Information hiding
    - Avoid direct access to data member
    - You must use Set and Get functions to access the data
  - Easy to use with validity check

```
1  // Fig. 9.16: Time.h
2  // Time class containing a constructor with default arguments.
3  // Member functions defined in Time.cpp.
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Time abstract data type definition
10 class Time
11 {
12 public:
13    Time( int = 0, int = 0, int = 0 ); // default constructor
14
15    // set functions
16    void setTime( int, int, int ); // set hour, minute, second
17    void setHour( int ); // set hour (after validation)
18    void setMinute( int ); // set minute (after validation)
19    void setSecond( int ); // set second (after validation)
20
```

Fig. 9.16 | `Time` class containing a constructor with default arguments. (Part 1 of 2.)

```
21      // get functions
22      int getHour(); // return hour
23      int getMinute(); // return minute
24      int getSecond(); // return second
25
26      void printUniversal(); // output time in universal-time format
27      void printStandard(); // output time in standard-time format
28   private:
29      int hour; // 0 - 23 (24-hour clock format)
30      int minute; // 0 - 59
31      int second; // 0 - 59
32   }; // end class Time
33
34   #endif
```

Fig. 9.16 | `Time` class containing a constructor with default arguments. (Part 2 of 2.)

```
1  // Fig. 9.17: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  #include <iomanip>
5  #include "Time.h" // include definition of class Time from Time.h
6  using namespace std;
7
8  // Time constructor initializes each data member to zero;
9  // ensures that Time objects start in a consistent state
10 Time::Time( int hr, int min, int sec )
11 {
12    setTime( hr, min, sec ); // validate and set time
13 } // end Time constructor
14
15 // set new Time value using universal time; ensure that
16 // the data remains consistent by setting invalid values to zero
17 void Time::setTime( int h, int m, int s )
18 {
19    setHour( h ); // set private field hour
20    setMinute( m ); // set private field minute
21    setSecond( s ); // set private field second
22 } // end function setTime
```

Fig. 9.17 | `Time` class member-function definitions including a constructor that takes arguments. (Part 1 of 4.)

```
23
24    // set hour value
25    void Time::setHour( int h )
26    {
27       hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
28    } // end function setHour
29
30    // set minute value
31    void Time::setMinute( int m )
32    {
33       minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
34    } // end function setMinute
35
36    // set second value
37    void Time::setSecond( int s )
38    {
39       second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
40    } // end function setSecond
41
```

Fig. 9.17 | `Time` class member-function definitions including a constructor that takes arguments. (Part 2 of 4.)

```
42   // return hour value
43   int Time::getHour()
44   {
45      return hour;
46   } // end function getHour
47
48   // return minute value
49   int Time::getMinute()
50   {
51      return minute;
52   } // end function getMinute
53
54   // return second value
55   int Time::getSecond()
56   {
57      return second;
58   } // end function getSecond
59
```

Fig. 9.17 | `Time` class member-function definitions including a constructor that takes arguments. (Part 3 of 4.)

```
60  // print Time in universal-time format (HH:MM:SS)
61  void Time::printUniversal()
62  {
63     cout << setfill( '0' ) << setw( 2 ) << getHour() << ":"
64        << setw( 2 ) << getMinute() << ":" << setw( 2 ) << getSecond();
65  } // end function printUniversal
66
67  // print Time in standard-time format (HH:MM:SS AM or PM)
68  void Time::printStandard()
69  {
70     cout << ( ( getHour() == 0 || getHour() == 12 ) ? 12 : getHour() % 12 )
71        << ":" << setfill( '0' ) << setw( 2 ) << getMinute()
72        << ":" << setw( 2 ) << getSecond() << ( hour < 12 ? " AM" : " PM" );
73  } // end function printStandard
```

Fig. 9.17 | `Time` class member-function definitions including a constructor that takes arguments. (Part 4 of 4.)

```
1  // Fig. 9.18: fig09_18.cpp
2  // Demonstrating the Time class set and get functions
3  #include <iostream>
4  #include "time.h"
5  using namespace std;
6
7  void incrementMinutes( Time &, const int ); // prototype
8
9  int main()
10 {
11    Time t; // create Time object
12
13    // set time using individual set functions
14    t.setHour( 17 );  // set hour to valid value
15    t.setMinute( 34 ); // set minute to valid value
16    t.setSecond( 25 ); // set second to valid value
17
18    // use get functions to obtain hour, minute and second
19    cout << "Result of setting all valid values:\n"
20         << "  Hour: " << t.getHour()
21         << "  Minute: " << t.getMinute()
22         << "  Second: " << t.getSecond();
```

Fig. 9.18 | *Set* and *get* functions manipulating an object's `private` data. (Part 1 of 4.)

```
23
24      // set time using individual set functions
25      t.setHour( 234 ); // invalid hour set to 0
26      t.setMinute( 43 ); // set minute to valid value
27      t.setSecond( 6373 ); // invalid second set to 0
28
29      // display hour, minute and second after setting
30      // invalid hour and second values
31      cout << "\n\nResult of attempting to set invalid hour and"
32           << " second:\n  Hour: " << t.getHour()
33           << "  Minute: " << t.getMinute()
34           << "  Second: " << t.getSecond() << "\n\n";
35
36      t.setTime( 11, 58, 0 ); // set time
37      incrementMinutes( t, 3 ); // increment t's minute by 3
38  } // end main
39
```

Fig. 9.18 | *Set* and *get* functions manipulating an object's `private` data. (Part 2 of 4.)

```
40  // add specified number of minutes to a Time object
41  void incrementMinutes( Time &tt, const int count )
42  {
43     cout << "Incrementing minute " << count
44          << " times:\nStart time: ";
45     tt.printStandard();
46
47     for ( int i = 0; i < count; i++ ) {
48        tt.setMinute( ( tt.getMinute() + 1 ) % 60 );
49
50        if ( tt.getMinute() == 0 )
51           tt.setHour( ( tt.getHour() + 1 ) % 24);
52
53        cout << "\nminute + 1: ";
54        tt.printStandard();
55     } // end for
56
57     cout << endl;
58  } // end function incrementMinutes
```

Fig. 9.18 | *Set* and *get* functions manipulating an object's `private` data. (Part 3 of 4.)

9.13 Default member-wise assignment

- =
  - Assign each data member from object 1 to the corresponding data member in object 2

```
1  // Fig. 9.19: Date.h
2  // Date class declaration.  Member functions are defined in Date.cpp.
3
4  // prevent multiple inclusions of header file
5  #ifndef DATE_H
6  #define DATE_H
7
8  // class Date definition
9  class Date
10 {
11 public:
12    Date( int = 1, int = 1, int = 2000 ); // default constructor
13    void print();
14 private:
15    int month;
16    int day;
17    int year;
18 }; // end class Date
19
20 #endif
```

Fig. 9.22 | `Date` class declaration.

```
1  // Fig. 9.20: 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  {
10    month = m;
11    day = d;
12    year = y;
13 } // end constructor Date
14
15 // print Date in the format mm/dd/yyyy
16 void Date::print()
17 {
18    cout << month << '/' << day << '/' << year;
19 } // end function print
```

Fig. 9.23 | `Date` class member-function definitions.

```
1  // Fig. 9.21: fig09_21.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. 9.24 | Default memberwise assignment. (Part 1 of 2.)

9.12 do not return a reference to a private data member

- Protect data member with private access
  - Usually apply Get and Set function to access private data member
  - Could be integrated with validity check
- Returning a reference may bypass the Get and Set function
  - Possibly bypass validity check

```
1  // Fig. 9.19: Time.h
2  // Time class declaration.
3  // Member functions defined in Time.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  class Time
10 {
11 public:
12    Time( int = 0, int = 0, int = 0 );
13    void setTime( int, int, int );
14    int getHour();
15    int &badSetHour( int ); // DANGEROUS reference return
16 private:
17    int hour;
18    int minute;
19    int second;
20 }; // end class Time
21
22 #endif
```

Fig. 9.19 | Time class declaration.

```
1  // Fig. 9.20: Time.cpp
2  // Time class member-function definitions.
3  #include "Time.h" // include definition of class Time
4
5  // constructor function to initialize private data; calls member function
6  // setTime to set variables; default values are 0 (see class definition)
7  Time::Time( int hr, int min, int sec )
8  {
9     setTime( hr, min, sec );
10 } // end Time constructor
11
12 // set values of hour, minute and second
13 void Time::setTime( int h, int m, int s )
14 {
15    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
16    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
17    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
18 } // end function setTime
19
20 // return hour value
21 int Time::getHour()
22 {
23    return hour;
24 } // end function getHour
```

Fig. 9.20 | Time class member-function definitions. (Part 1 of 2.)

```
25
26 // POOR PRACTICE: Returning a reference to a private data member.
27 int &Time::badSetHour( int hh )
28 {
29    hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
30    return hour; // DANGEROUS reference return
31 } // end function badSetHour
```

Fig. 9.20 | `Time` class member-function definitions. (Part 2 of 2.)

```
1  // Fig. 9.21: fig09_21.cpp
2  // Demonstrating a public member function that
3  // returns a reference to a private data member.
4  #include <iostream>
5  #include "Time.h" // include definition of class Time
6  using namespace std;
7
8  int main()
9  {
10    Time t; // create Time object
11
12    // initialize hourRef with the reference returned by badSetHour
13    int &hourRef = t.badSetHour( 20 ); // 20 is a valid hour
14
15    cout << "Valid hour before modification: " << hourRef;
16    hourRef = 30; // use hourRef to set invalid value in Time object t
17    cout << "\nInvalid hour after modification: " << t.getHour();
18
19    // Dangerous: Function call that returns
20    // a reference can be used as an lvalue!
21    t.badSetHour( 12 ) = 74; // assign another invalid value to hour
22
```

Fig. 9.21 | Returning a reference to a `private` data member. (Part 1 of 2.)

```
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. 9.21 | Returning a reference to a `private` data member. (Part 2 of 2.)

Summary: Chapter 9

- Object and Class
  - Data member
  - Member function
  - .
  - ->
  - Constructor
  - Destructor
- Separating interface from implementation
  - 3 files (main.cpp, myClass.h, myClass.cpp)
  - Complier and linker