Illustrations List *(Main Page)*

Fig. 6.1 Creating a structure, setting its members, and printing the structure.
Fig. 6.2 Simple definition of `class Time`.
Fig. 6.3 Abstract data type `Time` implementation as a class.
Fig. 6.4 Accessing an object's data members and member functions through each type of object handle—through the object's name, through a reference, and through a pointer to the object.
Fig. 6.5 Separating `Time` class interface and implementation.
Fig. 6.6 Erroneous attempt to access `private` members of a class.
Fig. 6.7 Using a utility function.
Fig. 6.8 Using a constructor with default arguments.
Fig. 6.9 Demonstrating the order in which constructors and destructors are called.
Fig. 6.10 Using set and get functions.
Fig. 6.11 Returning a reference to a private data member.

```
1  // Fig. 6.1: fig06_01.cpp
2  // Create a structure, set its members, and print it.
3  #include <iostream.h>
4
5  struct Time {     // structure definition
6     int hour;      // 0-23
7     int minute;    // 0-59
8     int second;    // 0-59
9  };
10
11 void printMilitary( const Time & );  // prototype
12 void printStandard( const Time & );  // prototype
13
14 int main()
15 {
16    Time dinnerTime;    // variable of new type Time
17
18    // set members to valid values
19    dinnerTime.hour = 18;
20    dinnerTime.minute = 30;
21    dinnerTime.second = 0;
```

Fig. 6.1 Creating a structure, setting its members, and printing the structure (part 1 of 2).

```
22
23    cout << "Dinner will be held at ";
24    printMilitary( dinnerTime );
25    cout << " military time,\nwhich is ";
26    printStandard( dinnerTime );
27    cout << " standard time.\n";
28
29    // set members to invalid values
30    dinnerTime.hour = 29;
31    dinnerTime.minute = 73;
32
33    cout << "\nTime with invalid values: ";
34    printMilitary( dinnerTime );
35    cout << endl;
36    return 0;
37 }
38
39 // Print the time in military format
40 void printMilitary( const Time &t )
41 {
42    cout << ( t.hour < 10 ? "0" : "" ) << t.hour << ":"
43         << ( t.minute < 10 ? "0" : "" ) << t.minute;
44 }
45
46 // Print the time in standard format
47 void printStandard( const Time &t )
48 {
49    cout << ( ( t.hour == 0 || t.hour == 12 ) ?
50              12 : t.hour % 12 )
51         << ":" << ( t.minute < 10 ? "0" : "" ) << t.minute
52         << ":" << ( t.second < 10 ? "0" : "" ) << t.second
53         << ( t.hour < 12 ? " AM" : " PM" );
54 }
```

```
Dinner will be held at 18:30 military time,
which is 6:30:00 PM standard time.

Time with invalid values: 29:73
```

Fig. 6.1 Creating a structure, setting its members, and printing the structure (part 2 of 2).

```
1  class Time {
2  public:
3     Time();
4     void setTime( int, int, int );
5     void printMilitary();
6     void printStandard();
7  private:
8     int hour;     // 0 - 23
9     int minute;   // 0 - 59
10    int second;   // 0 - 59
11 };
```

Fig. 6.2 Simple definition of `class Time`.

```
1  // Fig. 6.3: fig06_03.cpp
2  // Time class.
3  #include <iostream.h>
4
5  // Time abstract data type (ADT) definition
6  class Time {
7  public:
8     Time();                         // constructor
9     void setTime( int, int, int ); // set hour, minute, second
10    void printMilitary();          // print military time format
11    void printStandard();          // print standard time format
12 private:
13    int hour;     // 0 - 23
14    int minute;   // 0 - 59
15    int second;   // 0 - 59
16 };
17
18 // Time constructor initializes each data member to zero.
19 // Ensures all Time objects start in a consistent state.
20 Time::Time() { hour = minute = second = 0; }
```

Fig. 6.3 Abstract data type `Time` implementation as a class (part 1 of 3).

```
21
22 // Set a new Time value using military time. Perform validity
23 // checks on the data values. Set invalid values to zero.
24 void Time::setTime( int h, int m, int s )
25 {
26    hour = ( h >= 0 && h < 24 ) ? h : 0;
27    minute = ( m >= 0 && m < 60 ) ? m : 0;
28    second = ( s >= 0 && s < 60 ) ? s : 0;
29 }
30
31 // Print Time in military format
```

```
32  void Time::printMilitary()
33  {
34     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
35          << ( minute < 10 ? "0" : "" ) << minute;
36  }
37
38  // Print Time in standard format
39  void Time::printStandard()
40  {
41     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
42          << ":" << ( minute < 10 ? "0" : "" ) << minute
43          << ":" << ( second < 10 ? "0" : "" ) << second
44          << ( hour < 12 ? " AM" : " PM" );
45  }
46
47  // Driver to test simple class Time
48  int main()
49  {
50     Time t;  // instantiate object t of class Time
51
52     cout << "The initial military time is ";
53     t.printMilitary();
54     cout << "\nThe initial standard time is ";
55     t.printStandard();
56
57     t.setTime( 13, 27, 6 );
58     cout << "\n\nMilitary time after setTime is ";
59     t.printMilitary();
60     cout << "\nStandard time after setTime is ";
61     t.printStandard();
62
63     t.setTime( 99, 99, 99 );  // attempt invalid settings
64     cout << "\n\nAfter attempting invalid settings:"
65          << "\nMilitary time: ";
66     t.printMilitary();
67     cout << "\nStandard time: ";
68     t.printStandard();
69     cout << endl;
70     return 0;
71  }
```

Fig. 6.3 Abstract data type `Time` implementation as a class (part 2 of 3).

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

Fig. 6.3 Abstract data type `Time` implementation as a class (part 3 of 3).

```
1  // Fig. 6.4: fig06_04.cpp
2  // Demonstrating the class member access operators . and ->
3  //
4  // CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
5  #include <iostream.h>
6
7  // Simple class Count
8  class Count {
9  public:
10    int x;
11    void print() { cout << x << endl; }
12 };
13
14 int main()
15 {
16    Count counter,                // create counter object
17          *counterPtr = &counter, // pointer to counter
18          &counterRef = counter;  // reference to counter
19
20    cout << "Assign 7 to x and print using the object's name: ";
21    counter.x = 7;       // assign 7 to data member x
22    counter.print();     // call member function print
23
24    cout << "Assign 8 to x and print using a reference: ";
25    counterRef.x = 8;    // assign 8 to data member x
26    counterRef.print();  // call member function print
27
28    cout << "Assign 10 to x and print using a pointer: ";
29    counterPtr->x = 10;  // assign 10 to data member x
30    counterPtr->print(); // call member function print
31    return 0;
32 }
```

```
Assign 7 to x and print using the object's name: 7
Assign 8 to x and print using a reference: 8
Assign 10 to x and print using a pointer: 10
```

Fig. 6.4 Accessing an object's data members and member functions through each type of object handle—through the object's name, through a reference, and through a pointer to the object.

```
1  // Fig. 6.5: time1.h
2  // Declaration of the Time class.
3  // Member functions are defined in time1.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME1_H
7  #define TIME1_H
8
9  // Time abstract data type definition
10 class Time {
11 public:
12    Time();                         // constructor
13    void setTime( int, int, int ); // set hour, minute, second
14    void printMilitary();          // print military time format
15    void printStandard();          // print standard time format
16 private:
17    int hour;     // 0 - 23
18    int minute;   // 0 - 59
19    int second;   // 0 - 59
20 };
21
22 #endif
```

Fig. 6.5 Separating **Time** class interface and implementation (part 1 of 5).

```
23 // Fig. 6.5: time1.cpp
24 // Member function definitions for Time class.
25 #include <iostream.h>
26 #include "time1.h"
27
28 // Time constructor initializes each data member to zero.
29 // Ensures all Time objects start in a consistent state.
30 Time::Time() { hour = minute = second = 0; }
31
32 // Set a new Time value using military time. Perform validity
33 // checks on the data values. Set invalid values to zero.
34 void Time::setTime( int h, int m, int s )
35 {
36    hour   = ( h >= 0 && h < 24 ) ? h : 0;
37    minute = ( m >= 0 && m < 60 ) ? m : 0;
38    second = ( s >= 0 && s < 60 ) ? s : 0;
39 }
40
41 // Print Time in military format
42 void Time::printMilitary()
43 {
44    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
45         << ( minute < 10 ? "0" : "" ) << minute;
46 }
47
```

Fig. 6.5 Separating **Time** class interface and implementation (part 2 of 5).

```
48 // Print time in standard format
49 void Time::printStandard()
50 {
51    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
52         << ":" << ( minute < 10 ? "0" : "" ) << minute
53         << ":" << ( second < 10 ? "0" : "" ) << second
54         << ( hour < 12 ? " AM" : " PM" );
55 }
```

Fig. 6.5 Separating **Time** class interface and implementation (part 3 of 5).

```
// Fig. 6.5: fig06_05.cpp
// Driver for Time1 class
// NOTE: Compile with time1.cpp
#include <iostream.h>
#include "time1.h"

// Driver to test simple class Time
int main()
{
   Time t;  // instantiate object t of class time

   cout << "The initial military time is ";
   t.printMilitary();
   cout << "\nThe initial standard time is ";
   t.printStandard();

   t.setTime( 13, 27, 6 );
   cout << "\n\nMilitary time after setTime is ";
   t.printMilitary();
   cout << "\nStandard time after setTime is ";
   t.printStandard();

   t.setTime( 99, 99, 99 );  // attempt invalid settings
   cout << "\n\nAfter attempting invalid settings:\n"
        << "Military time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
}
```

Fig. 6.5 Separating `Time` class interface and implementation (part 4 of 5).

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM
Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

Fig. 6.5 Separating `Time` class interface and implementation (part 5 of 5).

```
1  // Fig. 6.6: fig06_06.cpp
2  // Demonstrate errors resulting from attempts
3  // to access private class members.
4  #include <iostream.h>
5  #include "time1.h"
6
7  int main()
8  {
9     Time t;
10
11    // Error: 'Time::hour' is not accessible
12    t.hour = 7;
13
14    // Error: 'Time::minute' is not accessible
15    cout << "minute = " << t.minute;
16
17    return 0;
18 }
```

```
Compiling FIG06_06.CPP:
Error FIG06_06.CPP 12: 'Time::hour' is not accessible
Error FIG06_06.CPP 15: 'Time::minute' is not accessible
```

Fig. 6.6 Erroneous attempt to access **`private`** members of a class.

```
1  // Fig. 6.7: salesp.h
2  // SalesPerson class definition
3  // Member functions defined in salesp.cpp
4  #ifndef SALESP_H
5  #define SALESP_H
6
7  class SalesPerson {
8  public:
9     SalesPerson();                // constructor
10    void getSalesFromUser(); // get sales figures from keyboard
11    void setSales( int, double ); // User supplies one month's
12                                  // sales figures.
13    void printAnnualSales();
14
15 private:
16    double totalAnnualSales(); // utility function
17    double sales[ 12 ];           // 12 monthly sales figures
18 };
19
20 #endif
```

Fig. 6.7 Using a utility function (part 1 of 5).

```
21 // Fig. 6.7: salesp.cpp
22 // Member functions for class SalesPerson
23 #include <iostream.h>
24 #include <iomanip.h>
25 #include "salesp.h"
26
27 // Constructor function initializes array
28 SalesPerson::SalesPerson()
29 {
30    for ( int i = 0; i < 12; i++ )
31       sales[ i ] = 0.0;
32 }
33
34 // Function to get 12 sales figures from the user
35 // at the keyboard
36 void SalesPerson::getSalesFromUser()
37 {
38    double salesFigure;
39
40    for ( int i = 0; i < 12; i++ ) {
41       cout << "Enter sales amount for month "
42            << i + 1 << ": ";
43       cin >> salesFigure;
44       setSales( i, salesFigure );
45    }
46 }
47
48 // Function to set one of the 12 monthly sales figures.
49 // Note that the month value must be from 0 to 11.
50 void SalesPerson::setSales( int month, double amount )
51 {
52    if ( month >= 0 && month < 12 && amount > 0 )
53       sales[ month ] = amount;
54    else
55       cout << "Invalid month or sales figure" << endl;
56 }
57
```

Fig. 6.7 Using a utility function (part 2 of 5).

```
58 // Print the total annual sales
59 void SalesPerson::printAnnualSales()
60 {
61    cout << setprecision( 2 )
62         << setiosflags( ios::fixed | ios::showpoint )
63         << "\nThe total annual sales are: $"
64         << totalAnnualSales() << endl;
65 }
66
67 // Private utility function to total annual sales
68 double SalesPerson::totalAnnualSales()
69 {
70    double total = 0.0;
71
72    for ( int i = 0; i < 12; i++ )
73       total += sales[ i ];
74
75    return total;
76 }
```

Fig. 6.7 Using a utility function (part 3 of 5).

```
// Fig. 6.7: fig06_07.cpp
// Demonstrating a utility function
// Compile with salesp.cpp
#include "salesp.h"

int main()
{
   SalesPerson s;         // create SalesPerson object s

   s.getSalesFromUser();  // note simple sequential code
   s.printAnnualSales();  // no control structures in main
   return 0;
}
```

Fig. 6.7 Using a utility function (part 4 of 5).

```
Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92

The total annual sales are: $60120.58
```

Fig. 6.7 Using a utility function (part 5 of 5).

```
// Fig. 6.8: time2.h
// Declaration of the Time class.
// Member functions are defined in time2.cpp

// preprocessor directives that
// prevent multiple inclusions of header file
#ifndef TIME2_H
#define TIME2_H

// Time abstract data type definition
class Time {
public:
   Time( int = 0, int = 0, int = 0 );  // default constructor
   void setTime( int, int, int ); // set hour, minute, second
   void printMilitary();          // print military time format
   void printStandard();          // print standard time format
private:
   int hour;     // 0 - 23
   int minute;   // 0 - 59
   int second;   // 0 - 59
};

#endif
```

Fig. 6.8 Using a constructor with default arguments (part 1 of 6).

```
24 // Fig. 6.8: time2.cpp
25 // Member function definitions for Time class.
26 #include <iostream.h>
27 #include "time2.h"
28
29 // Time constructor initializes each data member to zero.
30 // Ensures all Time objects start in a consistent state.
31 Time::Time( int hr, int min, int sec )
32    { setTime( hr, min, sec ); }
33
```

Fig. 6.8 Using a constructor with default arguments (part 2 of 6).

```
34 // Set a new Time value using military time. Perform validity
35 // checks on the data values. Set invalid values to zero.
36 void Time::setTime( int h, int m, int s )
37 {
38    hour   = ( h >= 0 && h < 24 ) ? h : 0;
39    minute = ( m >= 0 && m < 60 ) ? m : 0;
40    second = ( s >= 0 && s < 60 ) ? s : 0;
41 }
42
43 // Print Time in military format
44 void Time::printMilitary()
45 {
46    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
47         << ( minute < 10 ? "0" : "" ) << minute;
48 }
49
50 // Print Time in standard format
51 void Time::printStandard()
52 {
53    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
54         << ":" << ( minute < 10 ? "0" : "" ) << minute
55         << ":" << ( second < 10 ? "0" : "" ) << second
56         << ( hour < 12 ? " AM" : " PM" );
57 }
```

Fig. 6.8 Using a constructor with default arguments (part 3 of 6).

```
58 // Fig. 6.8: fig06_08.cpp
59 // Demonstrating a default constructor
60 // function for class Time.
61 #include <iostream.h>
62 #include "time2.h"
63
64 int main()
65 {
66    Time t1,             // all arguments defaulted
67         t2(2),          // minute and second defaulted
68         t3(21, 34),     // second defaulted
69         t4(12, 25, 42), // all values specified
70         t5(27, 74, 99); // all bad values specified
71
72    cout << "Constructed with:\n"
73         << "all arguments defaulted:\n   ";
74    t1.printMilitary();
75    cout << "\n   ";
76    t1.printStandard();
77
78    cout << "\nhour specified; minute and second defaulted:"
```

```
79          << "\n   ";
```

Fig. 6.8 Using a constructor with default arguments (part 4 of 6).

```
80     t2.printMilitary();
81     cout << "\n   ";
82     t2.printStandard();
83
84     cout << "\nhour and minute specified; second defaulted:"
85          << "\n   ";
86     t3.printMilitary();
87     cout << "\n   ";
88     t3.printStandard();
89
90     cout << "\nhour, minute, and second specified:"
91          << "\n   ";
92     t4.printMilitary();
93     cout << "\n   ";
94     t4.printStandard();
95
96     cout << "\nall invalid values specified:"
97          << "\n   ";
98     t5.printMilitary();
99     cout << "\n   ";
100    t5.printStandard();
101    cout << endl;
102
103    return 0;
104 }
```

Fig. 6.8 Using a constructor with default arguments (part 5 of 6).

```
Constructed with:
all arguments defaulted:
   00:00
   12:00:00 AM
hour specified; minute and second defaulted:
   02:00
   2:00:00 AM
hour and minute specified; second defaulted:
   21:34
   9:34:00 PM
hour, minute, and second specified:
   12:25
   12:25:42 PM
all invalid values specified:
   00:00
   12:00:00 AM
```

Fig. 6.8 Using a constructor with default arguments (part 6 of 6).

```
// Fig. 6.9: create.h
// Definition of class CreateAndDestroy.
// Member functions defined in create.cpp.
#ifndef CREATE_H
#define CREATE_H

class CreateAndDestroy {
public:
   CreateAndDestroy( int );  // constructor
   ~CreateAndDestroy();      // destructor
private:
   int data;
};

#endif
```

Fig. 6.9 Demonstrating the order in which constructors and destructors are called (part 1 of 4).

```
// Fig. 6.9: create.cpp
// Member function definitions for class CreateAndDestroy
#include <iostream.h>
#include "create.h"

CreateAndDestroy::CreateAndDestroy( int value )
{
   data = value;
   cout << "Object " << data << "   constructor";
}

CreateAndDestroy::~CreateAndDestroy()
   { cout << "Object " << data << "   destructor " << endl; }
```

Fig. 6.9 Demonstrating the order in which constructors and destructors are called (part 2 of 4).

```
// Fig. 6.9: fig06_09.cpp
// Demonstrating the order in which constructors and
// destructors are called.
#include <iostream.h>
#include "create.h"

void create( void );   // prototype

CreateAndDestroy first( 1 );  // global object

int main()
{
   cout << "   (global created before main)" << endl;
```

Fig. 6.9 Demonstrating the order in which constructors and destructors are called (part 3 of 4).

```

   CreateAndDestroy second( 2 );        // local object
   cout << "   (local automatic in main)" << endl;

   static CreateAndDestroy third( 3 );  // local object
   cout << "   (local static in main)" << endl;

   create();  // call function to create objects

   CreateAndDestroy fourth( 4 );        // local object
   cout << "   (local automatic in main)" << endl;
   return 0;
}
```

```
55
56 // Function to create objects
57 void create( void )
58 {
59    CreateAndDestroy fifth( 5 );
60    cout << "   (local automatic in create)" << endl;
61
62    static CreateAndDestroy sixth( 6 );
63    cout << "   (local static in create)" << endl;
64
65    CreateAndDestroy seventh( 7 );
66    cout << "   (local automatic in create)" << endl;
67 }
```

```
Object 1   constructor   (global created before main)
Object 2   constructor   (local automatic in main)
Object 3   constructor   (local static in main)
Object 5   constructor   (local automatic in create)
Object 6   constructor   (local static in create)
Object 7   constructor   (local automatic in create)
Object 7   destructor
Object 5   destructor
Object 4   constructor   (local automatic in main)
Object 4   destructor
Object 2   destructor
Object 6   destructor
Object 3   destructor
Object 1   destructor
```

Fig. 6.9 Demonstrating the order in which constructors and destructors are called (part 4 of 4).

```
1  // Fig. 6.10: time3.h
2  // Declaration of the Time class.
3  // Member functions defined in time3.cpp
4
5  // preprocessor directives that
6  // prevent multiple inclusions of header file
7  #ifndef TIME3_H
8  #define TIME3_H
9
10 class Time {
11 public:
12    Time( int = 0, int = 0, int = 0 );  // constructor
13
14    // set functions
15    void setTime( int, int, int );  // set hour, minute, second
16    void setHour( int );   // set hour
17    void setMinute( int ); // set minute
18    void setSecond( int ); // set second
```

Fig. 6.10 Using set and get functions (part 1 of 6).

```
19
20     // get functions
21     int getHour();         // return hour
22     int getMinute();       // return minute
23     int getSecond();       // return second
24
25     void printMilitary();  // output military time
26     void printStandard();  // output standard time
27
28  private:
29     int hour;              // 0 - 23
30     int minute;            // 0 - 59
31     int second;            // 0 - 59
32  };
33
34  #endif
```

Fig. 6.10 Using set and get functions (part 2 of 6).

```
35  // Fig. 6.10: time3.cpp
36  // Member function definitions for Time class.
37  #include "time3.h"
38  #include <iostream.h>
39
40  // Constructor function to initialize private data.
41  // Calls member function setTime to set variables.
42  // Default values are 0 (see class definition).
43  Time::Time( int hr, int min, int sec )
44     { setTime( hr, min, sec ); }
45
46  // Set the values of hour, minute, and second.
47  void Time::setTime( int h, int m, int s )
48  {
49     setHour( h );
50     setMinute( m );
51     setSecond( s );
52  }
53
54  // Set the hour value
55  void Time::setHour( int h )
56     { hour = ( h >= 0 && h < 24 ) ? h : 0; }
57
58  // Set the minute value
59  void Time::setMinute( int m )
60     { minute = ( m >= 0 && m < 60 ) ? m : 0; }
61
62  // Set the second value
63  void Time::setSecond( int s )
64     { second = ( s >= 0 && s < 60 ) ? s : 0; }
65
```

Fig. 6.10 Using set and get functions (part 3 of 6).

```
66  // Get the hour value
67  int Time::getHour() { return hour; }
68
69  // Get the minute value
70  int Time::getMinute() { return minute; }
71
72  // Get the second value
73  int Time::getSecond() { return second; }
74
75  // Print time in military format
```

```
76  void Time::printMilitary()
77  {
78     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
79          << ( minute < 10 ? "0" : "" ) << minute;
80  }
81
82  // Print time in standard format
83  void Time::printStandard()
84  {
85     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
86          << ":" << ( minute < 10 ? "0" : "" ) << minute
87          << ":" << ( second < 10 ? "0" : "" ) << second
88          << ( hour < 12 ? " AM" : " PM" );
89  }
```

Fig. 6.10 Using set and get functions (part 4 of 6).

```
90  // Fig. 6.10: fig06_10.cpp
91  // Demonstrating the Time class set and get functions
92  #include <iostream.h>
93  #include "time3.h"
94
95  void incrementMinutes( Time &, const int );
96
97  int main()
98  {
99     Time t;
100
101    t.setHour( 17 );
102    t.setMinute( 34 );
103    t.setSecond( 25 );
104
105    cout << "Result of setting all valid values:\n"
106         << "  Hour: " << t.getHour()
107         << "  Minute: " << t.getMinute()
108         << "  Second: " << t.getSecond();
109
110    t.setHour( 234 );    // invalid hour set to 0
111    t.setMinute( 43 );
112    t.setSecond( 6373 ); // invalid second set to 0
```

Fig. 6.10 Using set and get functions (part 5 of 6).

```
113
114    cout << "\n\nResult of attempting to set invalid hour and"
115         << " second:\n  Hour: " << t.getHour()
116         << "  Minute: " << t.getMinute()
117         << "  Second: " << t.getSecond() << "\n\n";
118
119    t.setTime( 11, 58, 0 );
120    incrementMinutes( t, 3 );
121
122    return 0;
123 }
124
125 void incrementMinutes(Time &tt, const int count)
126 {
127    cout << "Incrementing minute " << count
128         << " times:\nStart time: ";
129    tt.printStandard();
130
131    for ( int i = 0; i < count; i++ ) {
132       tt.setMinute( ( tt.getMinute() + 1 ) % 60);
```

```
133
134       if ( tt.getMinute() == 0 )
135          tt.setHour( ( tt.getHour() + 1 ) % 24);
136
137       cout << "\nminute + 1: ";
138       tt.printStandard();
139    }
140
141    cout << endl;
142 }
```

```
Result of setting all valid values:
  Hour: 17  Minute: 34  Second: 25

Result of attempting to set invalid hour and second:
  Hour: 0  Minute: 43  Second: 0

Incrementing minute 3 times:
Start time: 11:58:00 AM
minute + 1: 11:59:00 AM
minute + 1: 12:00:00 PM
minute + 1: 12:01:00 PM
```

Fig. 6.10 Using set and get functions (part 6 of 6).

```
1  // Fig. 6.11: time4.h
2  // Declaration of the Time class.
3  // Member functions defined in time4.cpp
4
5  // preprocessor directives that
6  // prevent multiple inclusions of header file
7  #ifndef TIME4_H
8  #define TIME4_H
9
10 class Time {
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 };
21
22 #endif
```

Fig. 6.11 Returning a reference to a private data member (part 1 of 4).

```
23 // Fig. 6.11: time4.cpp
24 // Member function definitions for Time class.
25 #include "time4.h"
26 #include <iostream.h>
27
28 // Constructor function to initialize private data.
29 // Calls member function setTime to set variables.
30 // Default values are 0 (see class definition).
31 Time::Time( int hr, int min, int sec )
32    { setTime( hr, min, sec ); }
33
34 // Set the values of hour, minute, and second.
35 void Time::setTime( int h, int m, int s )
36 {
37    hour   = ( h >= 0 && h < 24 ) ? h : 0;
38    minute = ( m >= 0 && m < 60 ) ? m : 0;
39    second = ( s >= 0 && s < 60 ) ? s : 0;
40 }
41
42 // Get the hour value
43 int Time::getHour() { return hour; }
44
45 // POOR PROGRAMMING PRACTICE:
46 // Returning a reference to a private data member.
47 int &Time::badSetHour( int hh )
48 {
49    hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
50
51    return hour;  // DANGEROUS reference return
52 }
```

Fig. 6.11 Returning a reference to a private data member (part 2 of 4).

```
53 // Fig. 6.11: fig06_11.cpp
54 // Demonstrating a public member function that
55 // returns a reference to a private data member.
56 // Time class has been trimmed for this example.
57 #include <iostream.h>
58 #include "time4.h"
59
60 int main()
61 {
62    Time t;
63    int &hourRef = t.badSetHour( 20 );
64
65    cout << "Hour before modification: " << hourRef;
66    hourRef = 30;  // modification with invalid value
67    cout << "\nHour after modification: " << t.getHour();
68
```

Fig. 6.11 Returning a reference to a private data member (part 3 of 4).

```
69    // Dangerous: Function call that returns
70    // a reference can be used as an lvalue!
71    t.badSetHour(12) = 74;
72    cout << "\n\n*********************************\n"
73         << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
74         << "badSetHour as an lvalue, Hour: "
75         << t.getHour()
76         << "\n*********************************" << endl;
77
78    return 0;
79 }
```

```
Hour before modification: 20
Hour after modification: 30

*********************************
POOR PROGRAMMING PRACTICE!!!!!!!!
badSetHour as an lvalue, Hour: 74
*********************************
```

Fig. 6.11 Returning a reference to a **`private`** data member (part 4 of 4).