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.

```
// Fig. 6.1: fig06_01.cpp
// Create a structure, set its members, and print it.
#include <iostream.h>

struct Time {    // structure definition
   int hour;     // 0-23
   int minute;   // 0-59
   int second;   // 0-59
};

void printMilitary( const Time & );  // prototype
void printStandard( const Time & );  // prototype

int main()
{
   Time dinnerTime;    // variable of new type Time

   // set members to valid values
   dinnerTime.hour = 18;
   dinnerTime.minute = 30;
   dinnerTime.second = 0;
```

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

```

   cout << "Dinner will be held at ";
   printMilitary( dinnerTime );
   cout << " military time,\nwhich is ";
   printStandard( dinnerTime );
   cout << " standard time.\n";

   // set members to invalid values
   dinnerTime.hour = 29;
   dinnerTime.minute = 73;

   cout << "\nTime with invalid values: ";
   printMilitary( dinnerTime );
   cout << endl;
   return 0;
}

// Print the time in military format
void printMilitary( const Time &t )
{
   cout << ( t.hour < 10 ? "0" : "" ) << t.hour << ":"
        << ( t.minute < 10 ? "0" : "" ) << t.minute;
}

// Print the time in standard format
void printStandard( const Time &t )
{
   cout << ( ( t.hour == 0 || t.hour == 12 ) ?
             12 : t.hour % 12 )
        << ":" << ( t.minute < 10 ? "0" : "" ) << t.minute
        << ":" << ( t.second < 10 ? "0" : "" ) << t.second
        << ( t.hour < 12 ? " AM" : " PM" );
}
```

```
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).

```
class Time {
public:
   Time();
   void setTime( int, int, int );
   void printMilitary();
   void printStandard();
private:
   int hour;     // 0 - 23
   int minute;   // 0 - 59
   int second;   // 0 - 59
};
```

Fig. 6.2 Simple definition of `class Time`.

```
// Fig. 6.3: fig06_03.cpp
// Time class.
#include <iostream.h>

// Time abstract data type (ADT) definition
class Time {
public:
   Time();                         // 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
};

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time() { hour = minute = second = 0; }
```

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

```

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Print Time in military format
```

```
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}

// Print Time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}

// 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:"
        << "\nMilitary time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
}
```

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).

```
// Fig. 6.4: fig06_04.cpp
// Demonstrating the class member access operators . and ->
//
// CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
#include <iostream.h>

// Simple class Count
class Count {
public:
   int x;
   void print() { cout << x << endl; }
};

int main()
{
   Count counter,                 // create counter object
         *counterPtr = &counter, // pointer to counter
         &counterRef = counter;  // reference to counter

   cout << "Assign 7 to x and print using the object's name: ";
   counter.x = 7;       // assign 7 to data member x
   counter.print();     // call member function print

   cout << "Assign 8 to x and print using a reference: ";
   counterRef.x = 8;    // assign 8 to data member x
   counterRef.print();  // call member function print

   cout << "Assign 10 to x and print using a pointer: ";
   counterPtr->x = 10;  // assign 10 to data member x
   counterPtr->print(); // call member function print
   return 0;
}
```

```
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.

```
// Fig. 6.5: time1.h
// Declaration of the Time class.
// Member functions are defined in time1.cpp

// prevent multiple inclusions of header file
#ifndef TIME1_H
#define TIME1_H

// Time abstract data type definition
class Time {
public:
   Time();                          // 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. 6.5 Separating `Time` class interface and implementation (part 1 of 5).

```
// Fig. 6.5: time1.cpp
// Member function definitions for Time class.
#include <iostream.h>
#include "time1.h"

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time() { hour = minute = second = 0; }

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour   = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}

```

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

```
// Print time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```

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).

```cpp
// Fig. 6.6: fig06_06.cpp
// Demonstrate errors resulting from attempts
// to access private class members.
#include <iostream.h>
#include "time1.h"

int main()
{
   Time t;

   // Error: 'Time::hour' is not accessible
   t.hour = 7;

   // Error: 'Time::minute' is not accessible
   cout << "minute = " << t.minute;

   return 0;
}
```

```
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.

```cpp
// Fig. 6.7: salesp.h
// SalesPerson class definition
// Member functions defined in salesp.cpp
#ifndef SALESP_H
#define SALESP_H

class SalesPerson {
public:
   SalesPerson();                // constructor
   void getSalesFromUser(); // get sales figures from keyboard
   void setSales( int, double ); // User supplies one month's
                                 // sales figures.
   void printAnnualSales();

private:
   double totalAnnualSales(); // utility function
   double sales[ 12 ];           // 12 monthly sales figures
};

#endif
```

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

```
// Fig. 6.7: salesp.cpp
// Member functions for class SalesPerson
#include <iostream.h>
#include <iomanip.h>
#include "salesp.h"

// Constructor function initializes array
SalesPerson::SalesPerson()
{
   for ( int i = 0; i < 12; i++ )
      sales[ i ] = 0.0;
}

// Function to get 12 sales figures from the user
// at the keyboard
void SalesPerson::getSalesFromUser()
{
   double salesFigure;

   for ( int i = 0; i < 12; i++ ) {
      cout << "Enter sales amount for month "
           << i + 1 << ": ";
      cin >> salesFigure;
      setSales( i, salesFigure );
   }
}

// Function to set one of the 12 monthly sales figures.
// Note that the month value must be from 0 to 11.
void SalesPerson::setSales( int month, double amount )
{
   if ( month >= 0 && month < 12 && amount > 0 )
      sales[ month ] = amount;
   else
      cout << "Invalid month or sales figure" << endl;
}

```

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

```
// Print the total annual sales
void SalesPerson::printAnnualSales()
{
   cout << setprecision( 2 )
        << setiosflags( ios::fixed | ios::showpoint )
        << "\nThe total annual sales are: $"
        << totalAnnualSales() << endl;
}

// Private utility function to total annual sales
double SalesPerson::totalAnnualSales()
{
   double total = 0.0;

   for ( int i = 0; i < 12; i++ )
      total += sales[ i ];

   return total;
}
```

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).

```
// Fig. 6.8: time2.cpp
// Member function definitions for Time class.
#include <iostream.h>
#include "time2.h"

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time( int hr, int min, int sec )
   { setTime( hr, min, sec ); }

```

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

```
// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour   = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}

// Print Time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```

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

```
// Fig. 6.8: fig06_08.cpp
// Demonstrating a default constructor
// function for class Time.
#include <iostream.h>
#include "time2.h"

int main()
{
   Time t1,             // all arguments defaulted
        t2(2),          // minute and second defaulted
        t3(21, 34),     // second defaulted
        t4(12, 25, 42), // all values specified
        t5(27, 74, 99); // all bad values specified

   cout << "Constructed with:\n"
        << "all arguments defaulted:\n   ";
   t1.printMilitary();
   cout << "\n   ";
   t1.printStandard();

   cout << "\nhour specified; minute and second defaulted:"
```

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

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

```
   t2.printMilitary();
   cout << "\n   ";
   t2.printStandard();

   cout << "\nhour and minute specified; second defaulted:"
        << "\n   ";
   t3.printMilitary();
   cout << "\n   ";
   t3.printStandard();

   cout << "\nhour, minute, and second specified:"
        << "\n   ";
   t4.printMilitary();
   cout << "\n   ";
   t4.printStandard();

   cout << "\nall invalid values specified:"
        << "\n   ";
   t5.printMilitary();
   cout << "\n   ";
   t5.printStandard();
   cout << endl;

   return 0;
}
```

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;
}
```

```

// Function to create objects
void create( void )
{
   CreateAndDestroy fifth( 5 );
   cout << "   (local automatic in create)" << endl;

   static CreateAndDestroy sixth( 6 );
   cout << "   (local static in create)" << endl;

   CreateAndDestroy seventh( 7 );
   cout << "   (local automatic in create)" << endl;
}
```

```
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).

```
// Fig. 6.10: time3.h
// Declaration of the Time class.
// Member functions defined in time3.cpp

// preprocessor directives that
// prevent multiple inclusions of header file
#ifndef TIME3_H
#define TIME3_H

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

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

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

```

   // get functions
   int getHour();         // return hour
   int getMinute();       // return minute
   int getSecond();       // return second

   void printMilitary();  // output military time
   void printStandard();  // output standard time

private:
   int hour;              // 0 - 23
   int minute;            // 0 - 59
   int second;            // 0 - 59
};

#endif
```

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

```
// Fig. 6.10: time3.cpp
// Member function definitions for Time class.
#include "time3.h"
#include <iostream.h>

// Constructor function to initialize private data.
// Calls member function setTime to set variables.
// Default values are 0 (see class definition).
Time::Time( int hr, int min, int sec )
   { setTime( hr, min, sec ); }

// Set the values of hour, minute, and second.
void Time::setTime( int h, int m, int s )
{
   setHour( h );
   setMinute( m );
   setSecond( s );
}

// Set the hour value
void Time::setHour( int h )
   { hour = ( h >= 0 && h < 24 ) ? h : 0; }

// Set the minute value
void Time::setMinute( int m )
   { minute = ( m >= 0 && m < 60 ) ? m : 0; }

// Set the second value
void Time::setSecond( int s )
   { second = ( s >= 0 && s < 60 ) ? s : 0; }

```

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

```
// Get the hour value
int Time::getHour() { return hour; }

// Get the minute value
int Time::getMinute() { return minute; }

// Get the second value
int Time::getSecond() { return second; }

// Print time in military format
```

```
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}

// Print time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```

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

```
// Fig. 6.10: fig06_10.cpp
// Demonstrating the Time class set and get functions
#include <iostream.h>
#include "time3.h"

void incrementMinutes( Time &, const int );

int main()
{
   Time t;

   t.setHour( 17 );
   t.setMinute( 34 );
   t.setSecond( 25 );

   cout << "Result of setting all valid values:\n"
        << "  Hour: " << t.getHour()
        << "  Minute: " << t.getMinute()
        << "  Second: " << t.getSecond();

   t.setHour( 234 );    // invalid hour set to 0
   t.setMinute( 43 );
   t.setSecond( 6373 ); // invalid second set to 0
```

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

```

   cout << "\n\nResult of attempting to set invalid hour and"
        << " second:\n  Hour: " << t.getHour()
        << "  Minute: " << t.getMinute()
        << "  Second: " << t.getSecond() << "\n\n";

   t.setTime( 11, 58, 0 );
   incrementMinutes( t, 3 );

   return 0;
}

void incrementMinutes(Time &tt, const int count)
{
   cout << "Incrementing minute " << count
        << " times:\nStart time: ";
   tt.printStandard();

   for ( int i = 0; i < count; i++ ) {
      tt.setMinute( ( tt.getMinute() + 1 ) % 60);

      if ( tt.getMinute() == 0 )
         tt.setHour( ( tt.getHour() + 1 ) % 24);

      cout << "\nminute + 1: ";
      tt.printStandard();
   }

   cout << endl;
}
```

```
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).

```
// Fig. 6.11: time4.h
// Declaration of the Time class.
// Member functions defined in time4.cpp

// preprocessor directives that
// prevent multiple inclusions of header file
#ifndef TIME4_H
#define TIME4_H

class Time {
public:
   Time( int = 0, int = 0, int = 0 );
   void setTime( int, int, int );
   int getHour();
   int &badSetHour( int );  // DANGEROUS reference return
private:
   int hour;
   int minute;
   int second;
};

#endif
```

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

```
// Fig. 6.11: time4.cpp
// Member function definitions for Time class.
#include "time4.h"
#include <iostream.h>

// Constructor function to initialize private data.
// Calls member function setTime to set variables.
// Default values are 0 (see class definition).
Time::Time( int hr, int min, int sec )
   { setTime( hr, min, sec ); }

// Set the values of hour, minute, and second.
void Time::setTime( int h, int m, int s )
{
   hour   = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Get the hour value
int Time::getHour() { return hour; }

// POOR PROGRAMMING PRACTICE:
// Returning a reference to a private data member.
int &Time::badSetHour( int hh )
{
   hour = ( hh >= 0 && hh < 24 ) ? hh : 0;

   return hour;  // DANGEROUS reference return
}
```

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

```
// Fig. 6.11: fig06_11.cpp
// Demonstrating a public member function that
// returns a reference to a private data member.
// Time class has been trimmed for this example.
#include <iostream.h>
#include "time4.h"

int main()
{
   Time t;
   int &hourRef = t.badSetHour( 20 );

   cout << "Hour before modification: " << hourRef;
   hourRef = 30;  // modification with invalid value
   cout << "\nHour after modification: " << t.getHour();

```

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

```
   // Dangerous: Function call that returns
   // a reference can be used as an lvalue!
   t.badSetHour(12) = 74;
   cout << "\n\n*********************************\n"
        << "POOR PROGRAMMING PRACTICE!!!!!!!!!\n"
        << "badSetHour as an lvalue, Hour: "
        << t.getHour()
        << "\n*********************************" << endl;

   return 0;
}
```

```
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).