

# Chapter 16: Classes and Data Abstraction

## Outline

- 16.1 Introduction
- 16.2 Implementing a Time Abstract Data Type with a Class
- 16.3 Class Scope and Accessing Class Members
- 16.4 Separating Interface from Implementation
- 16.5 Controlling Access to Members
- 16.6 Access Functions and Utility Functions
- 16.7 Initializing Class Objects: Constructors
- 16.8 Using Default Arguments with Constructors
- 16.9 Using Destructors
- 16.10 When Constructors and Destructors Are Called
- 16.11 Using Data Members and Member Functions
- 16.12 A Subtle Trap: Returning a Reference to a `private` Data Member
- 16.13 Assignment by Default Memberwise Copy
- 16.14 Software Reusability



## 16.1 Introduction

- Object-oriented programming (OOP)
  - *Encapsulates* data (attributes) and functions (behavior) into packages called *classes*
  - Data and functions closely related
- Information hiding
  - Implementation details are hidden within the classes themselves
- Unit of C++ programming: the class
  - A class is like a blueprint – reusable
  - Objects are *instantiated* (created) from the class
  - For example, a house is an instance of a “blueprint class”
  - C programmers concentrate on functions



# 16.2 Implementing a Time Abstract Data Type with a Class

- Classes
  - Model objects that have attributes (data members) and behaviors (member functions)
  - Defined using keyword **class**

```
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 };
```

**Public:** and **Private:** are member-access specifiers.

**setTime**, **printMilitary**, and **printStandard** are member functions.  
**Time** is the constructor.

**hour**, **minute**, and **second** are data members.



## 16.2 Implementing a Time Abstract Data Type with a Class (II)

- Format
  - Body delineated with braces ( { and } )
  - Class definition terminates with a semicolon
- Member functions and data
  - Public** - accessible wherever the program has access to an object of class **Time**
  - Private** - accessible only to member functions of the class
  - Protected** - discussed later in the course



## 16.2 Implementing a Time Abstract Data Type with a Class (III)

- Constructor
  - Special member function that initializes data members of a class object
  - Constructors cannot return values
  - Same name as the class
- Declarations
  - Once class defined, can be used as a data type

```
Time sunset,           // object of type Time
    arrayOfTimes[ 5 ], // array of Time objects
    *pointerToTime,    // pointer to a Time object
    &dinnerTime = sunset; // reference to a Time object
```

Note: The class name becomes the new type specifier.



## 16.2 Implementing a Time Abstract Data Type with a Class (IV)

- Binary scope resolution operator (`::`)
  - Specifies which class owns the member function
  - Different classes can have the same name for member functions

- Format for definition class member functions

*ReturnType ClassName::MemberFunctionName()*{

*...*

*}*



## 16.2 Implementing a Time Abstract Data Type with a Class (V)

- If member function is defined *inside* the class
  - Scope resolution operator and class name are not needed
  - Defining a function outside a class does not change it being **public** or **private**
- Classes encourage software reuse
  - Inheritance allows new classes to be derived from old ones
- In following program
  - **Time** constructor initializes the data members to 0
    - Ensures that the object is in a consistent state when it is created





## 1. Class definition

### 1.1 Define default values

```
1 // Fig. 16.2: fig16_02.cpp
2 // Time class.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 // Time abstract data type (ADT) definition
9 class Time {
10 public:
11     Time(); // constructor
12     void setTime( int, int, int ); // set hour, minute, second
13     void printMilitary(); // print military time format
14     void printStandard(); // print standard time format
15 private:
16     int hour; // 0 - 23
17     int minute; // 0 - 59
18     int second; // 0 - 59
19 };
20
21 // Time constructor initializes each data member to zero.
22 // Ensures all Time objects start in a consistent state.
23 Time::Time() { hour = minute = second = 0; }
24
25 // Set a new Time value using military time. Perform validity
26 // checks on the data values. Set invalid values to zero.
27 void Time::setTime( int h, int m, int s )
28 {
29     hour = ( h >= 0 && h < 24 ) ? h : 0;
30     minute = ( m >= 0 && m < 60 ) ? m : 0;
31     second = ( s >= 0 && s < 60 ) ? s : 0;
32 }
```





## Outline

**1.2 Define the two functions  
printMilitary and  
printStandard**

**2. In main(), create  
an object of class  
Time.**

**2.1 Print the initial  
(default) time**

```
33
34 // Print Time in military format
35 void Time::printMilitary()
36 {
37     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
38         << ( minute < 10 ? "0" : "" ) << minute;
39 }
40
41 // Print Time in standard format
42 void Time::printStandard()
43 {
44     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
45         << ":" << ( minute < 10 ? "0" : "" ) << minute
46         << ":" << ( second < 10 ? "0" : "" ) << second
47         << ( hour < 12 ? " AM" : " PM" );
48 }
49
50 // Driver to test simple class Time
51 int main()
52 {
53     Time t; // instantiate object t of class Time
54
55     cout << "The initial military time is ";
56     t.printMilitary();
57     cout << "\nThe initial standard time is ";
58     t.printStandard();
59
```



## Outline



**2.2 Set and print the time.**

**2.3 Attempt to set the time to an invalid hour**

**2.4 Print**

## **Program Output**

```
60     t.setTime( 13, 27, 6 );
61     cout << "\n\nMilitary time after setTime is ";
62     t.printMilitary();
63     cout << "\nStandard time after setTime is ";
64     t.printStandard();
65
66     t.setTime( 99, 99, 99 ); // attempt invalid settings
67     cout << "\n\nAfter attempting invalid settings:"
68         << "\nMilitary time: ";
69     t.printMilitary();
70     cout << "\nStandard time: ";
71     t.printStandard();
72     cout << endl;
73     return 0;
74 }
```

```
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
```

## 16.3 Class Scope and Accessing Class Members

- Class scope
  - Data members and member functions
- File scope
  - Nonmember functions
- Function scope
  - Variables defined in member functions, destroyed after function completes
- Inside a scope
  - Members accessible by all member functions
  - Referenced by name



## 16.3 Class Scope and Accessing Class Members (II)

- Outside a scope
  - Use handles
    - An object name, a reference to an object or a pointer to an object
- Accessing class members
  - Same as **structs**
  - Dot (.) for objects and arrow (->) for pointers
  - Example: **t.hour** is the hour element of **t**
  - **TimePtr->hour** is the hour element





## Outline



### 1. Class definition

#### 1.1 Initialize object

#### 2. Print using the dot operator

#### 2.2 Set new value

#### 2.3 Print using a reference

```
1 // Fig. 16.3: fig16_03.cpp
2 // Demonstrating the class member access operators . and ->
3 //
4 // CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
5 #include <iostream>
6
7 using std::cout;
8 using std::endl;
9
10 // Simple class Count
11 class Count {
12 public:
13     int x;
14     void print() { cout << x << endl; }
15 };
16
17 int main()
18 {
19     Count counter,           // create counter object
20         *counterPtr = &counter, // pointer to counter
21         &counterRef = counter; // reference to counter
22
23     cout << "Assign 7 to x and print using the object's name: ";
24     counter.x = 7;           // assign 7 to data member x
25     counter.print();         // call member function print
26
27     cout << "Assign 8 to x and print using a reference: ";
28     counterRef.x = 8;        // assign 8 to data member x
29     counterRef.print();      // call member function print
30 }
```

```
31     cout << "Assign 10 to x and print using a pointer: ";  
32     counterPtr->x = 10;    // assign 10 to data member x  
33     counterPtr->print(); // call member function print  
34     return 0;  
35 }
```



## Outline



### 2.3 Set new value

### 2.4 Print using a pointer

```
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
```

## Program Output

## 16.4 Separating Interface from Implementation

- Separating interface from implementation
  - Easier to modify programs
  - C++ programs can be split into
    - Header files* – contains class definitions and function prototypes
    - Source-code files* – contains member function definitions
- Program Outline:
  - Using the same **Time** class as before, create a header file
  - Create a source code file
    - Load the header file to get the class definitions
    - Define the member functions of the class





## Outline



Header file (function prototypes, class definitions)

### 1. Class definition

```
1 // Fig. 16.4: 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
```





## Outline



### Source code file (function definitions)

#### 2.1 Load the header

#### 2.2. Define the member functions

```
23 // Fig. 16.4: time1.cpp
24 // Member function definitions for Time class.
25 #include <iostream>
26
27 using std::cout;
28
29 #include "time1.h"
30
31 // Time constructor initializes each data member to zero.
32 // Ensures all Time objects start in a consistent state.
33 Time::Time() { hour = minute = second = 0; }
34
35 // Set a new Time value using military time. Perform validity
36 // checks on the data values. Set invalid values to zero.
37 void Time::setTime( int h, int m, int s )
38 {
39     hour    = ( h >= 0 && h < 24 ) ? h : 0;
40     minute  = ( m >= 0 && m < 60 ) ? m : 0;
41     second  = ( s >= 0 && s < 60 ) ? s : 0;
42 }
43
44 // Print Time in military format
45 void Time::printMilitary()
46 {
47     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
48           << ( minute < 10 ? "0" : "" ) << minute;
49 }
```



## Outline



### **2.2. Define the member functions**

```
50
51 // Print time in standard format
52 void Time::printStandard()
53 {
54     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
55         << ":" << ( minute < 10 ? "0" : "" ) << minute
56         << ":" << ( second < 10 ? "0" : "" ) << second
57         << ( hour < 12 ? " AM" : " PM" );
58 }
```



## Outline



### 1. Load header

#### 1.1 Initialize object

### 2. Function calls

### 3. Print

```
59 // Fig. 16.4: fig16_04.cpp
60 // Driver for Time1 class
61 // NOTE: Compile with time1.cpp
62 #include <iostream>
63
64 using std::cout;
65 using std::endl;
66
67 #include "time1.h"
68
69 // Driver to test simple class Time
70 int main()
71 {
72     Time t; // instantiate object t of class time
73
74     cout << "The initial military time is ";
75     t.printMilitary();
76     cout << "\nThe initial standard time is ";
77     t.printStandard();
78
79     t.setTime( 13, 27, 6 );
80     cout << "\n\nMilitary time after setTime is ";
81     t.printMilitary();
82     cout << "\nStandard time after setTime is ";
83     t.printStandard();
84 }
```



```
85     t.setTime( 99, 99, 99 );  // attempt invalid settings
86     cout << "\n\nAfter attempting invalid settings:\n"
87         << "Military time: ";
88     t.printMilitary();
89     cout << "\nStandard time: ";
90     t.printStandard();
91     cout << endl;
92     return 0;
93 }
```

```
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
```

## Program Output

## 16.5 Controlling Access to Members

- Purpose of **public**
  - Give clients a view of the *services* the class provides (interface)
- Purpose of **private**
  - Default setting
  - Hide details of how the class accomplishes its tasks (implementation)
  - **Private** members only accessible through the **public** interface using **public** member functions





## Outline



1. Load header file for Time class.

2. Create an object of class Time.

2.1 Attempt to set a private variable

2.2 Attempt to access a private variable.

```
1 // Fig. 16.5: fig16_05.cpp
2 // Demonstrate errors resulting from attempts
3 // to access private class members.
4 #include <iostream>
5
6 using std::cout;
7
8 #include "time1.h"
9
10 int main()
11 {
12     Time t;
13
14     // Error: 'Time::hour' is not accessible
15     t.hour = 7;
16
17     // Error: 'Time::minute' is not accessible
18     cout << "minute = " << t.minute;
19
20     return 0;
21 }
```

Compiling...

Fig06\_06.cpp

D:\Fig06\_06.cpp(15) : error C2248: 'hour' : cannot access private member declared in class 'Time'

D:\Fig6\_06\time1.h(18) : see declaration of 'hour'

D:\Fig06\_06.cpp(18) : error C2248: 'minute' : cannot access private member declared in class 'Time'

D:\time1.h(19) : see declaration of 'minute'

Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)

## Program Output

## 16.6 Access Functions and Utility Functions

- Utility functions
  - **private** functions that support the operation of public functions
  - Not intended to be used directly by clients
- Access functions
  - **public** functions that read/display data or check conditions
  - For a container, it could call the **isEmpty** function
- Next
  - Program to take in monthly sales and output the total
  - Implementation not shown, only access functions





## Outline



**1. Load header file  
(compile with file that  
contains function  
definitions)**

**1.1 Create an object**

**2. Function calls**

**Program Output**

```
87 // Fig. 16.6: fig16_06.cpp
88 // Demonstrating a utility function
89 // Compile with salesp.cpp
90 #include "salesp.h"
91
92 int main()
93 {
94     SalesPerson s;           // create SalesPerson object s
95
96     s.getSalesFromUser();    // note simple sequential code
97     s.printAnnualSales();    // no control structures in main
98     return 0;
99 }
```

### OUTPUT

```
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: $60120.59
```



## 16.7 Initializing Class Objects: Constructors

- Constructor function
  - Can initialize class members
  - Same name as the class, no return type
  - Member variables can be initialized by the constructor or set afterwards
- Declaring objects
  - Initializers can be provided
  - Initializers passed as arguments to the class' constructor



## 16.7 Initializing Class Objects: Constructors (II)

- Format

*Type* *ObjectName* ( *value1*, *value2*, ...) ;

- Constructor assigns *value1*, *value2*, etc. to its member variables
- If not enough values specified, rightmost parameters set to their default (specified by programmer)

**myClass myObject( 3, 4.0 ) ;**



## 16.8 Using Default Arguments with Constructors

- Default constructor
  - One per class
  - Can be invoked without arguments
  - Has default arguments
- Default arguments
  - Set in default constructor function prototype (in class definition)
    - Do not set defaults in the function definition, outside of a class
  - Example:  
**`SampleClass( int = 0, float = 0) ;`**
    - Constructor has same name as class





## Outline



### 1. Define class Time and its default values.

```
1 // Fig. 16.7: time2.h
2 // Declaration of the Time class.
3 // Member functions are defined in time2.cpp
4
5 // preprocessor directives that
6 // prevent multiple inclusions of header file
7 #ifndef TIME2_H
8 #define TIME2_H
9
10 // Time abstract data type definition
11 class Time {
12 public:
13     Time( int = 0, int = 0, int = 0 ); // default constructor
14     void setTime( int, int, int ); // set hour, minute, second
15     void printMilitary();           // print military time format
16     void printStandard();           // print standard time format
17 private:
18     int hour;           // 0 - 23
19     int minute;         // 0 - 59
20     int second;         // 0 - 59
21 };
22
23 #endif
```



## Outline



### 2. Create objects using default arguments.

#### 2.1 Print the objects.

```
61 // Fig. 16.7: fig16_07.cpp
62 // Demonstrating a default constructor
63 // function for class Time.
64 #include <iostream>
65
66 using std::cout;
67 using std::endl;
68
69 #include "time2.h"
70
71 int main()
72 {
73     Time t1,           // all arguments defaulted
74         t2(2),         // minute and second defaulted
75         t3(21, 34),    // second defaulted
76         t4(12, 25, 42), // all values specified
77         t5(27, 74, 99); // all bad values specified
78
79     cout << "Constructed with:\n"
80          << "all arguments defaulted:\n    ";
81     t1.printMilitary();
82     cout << "\n    ";
83     t1.printStandard();
84
85     cout << "\nhour specified; minute and second defaulted:"
86          << "\n    ";
87     t2.printMilitary();
88     cout << "\n    ";
89     t2.printStandard();
90
91     cout << "\nhour and minute specified; second defaulted:"
92          << "\n    ";
93     t3.printMilitary();
```



## Outline



### 2.1 (continued) Print the objects.

```
94     cout << "\n    ";
95     t3.printStandard();
96
97     cout << "\nhour, minute, and second specified:"
98         << "\n    ";
99     t4.printMilitary();
100    cout << "\n    ";
101    t4.printStandard();
102
103    cout << "\nall invalid values specified:"
104        << "\n    ";
105    t5.printMilitary();
106    cout << "\n    ";
107    t5.printStandard();
108    cout << endl;
109
110    return 0;
111 }
```

```
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
```

### Program Output

## 16.9 Using Destructors

- Destructor
  - Member function of class
  - Performs termination housekeeping before the system reclaims the object's memory
  - Complement of the constructor
  - Name is *tilde* (~) followed by the class name
    - **~Time**
    - Recall that the constructor's name is the class name
  - Receives no parameters, returns no value
  - One destructor per class - no overloading allowed



## 16.10 When Constructors and Destructors Are Called

- Constructors and destructors called automatically
  - Order depends on scope of objects
- Global scope objects
  - Constructors called before any other function (including **main**)
  - Destructors called when **main** terminates (or **exit** function called)
  - Destructors not called if program terminates with **abort**





## 16.10 When Constructors and Destructors Are Called (II)

- Automatic local objects
  - Constructors called when objects defined
  - Destructors called when objects leave scope (when the block in which they are defined is exited)
  - Destructors not called if program ends with **exit** or **abort**
- **static** local objects
  - Constructors called when execution reaches the point where the objects are defined
  - Destructors called when **main** terminates or the **exit** function is called
  - Destructors not called if the program ends with **abort**





## Outline

### 1. Create header file

#### 1.1 Function prototypes

```
1 // Fig. 16.8: create.h
2 // Definition of class CreateAndDestroy.
3 // Member functions defined in create.cpp.
4 #ifndef CREATE_H
5 #define CREATE_H
6
7 class CreateAndDestroy {
8 public:
9     CreateAndDestroy( int ); // constructor
10    ~CreateAndDestroy();      // destructor
11 private:
12    int data;
13 };
14
15 #endif
```



## Outline



### 1. Load header

#### 1.1 Function definitions

```
16 // Fig. 16.8: create.cpp
17 // Member function definitions for class CreateAndDestroy
18 #include <iostream>
19
20 using std::cout;
21 using std::endl;
22
23 #include "create.h"
24
25 CreateAndDestroy::CreateAndDestroy( int value )
26 {
27     data = value;
28     cout << "Object " << data << "   constructor";
29 }
30
31 CreateAndDestroy::~~CreateAndDestroy()
32 { cout << "Object " << data << "   destructor " << endl; }
```



## Outline



### 1. Load header

#### 1.1 Initialize objects

### 2. Print

```
33 // Fig. 16.8: fig16_08.cpp
34 // Demonstrating the order in which constructors and
35 // destructors are called.
36 #include <iostream>
37
38 using std::cout;
39 using std::endl;
40
41 #include "create.h"
42
43 void create( void );    // prototype
44
45 CreateAndDestroy first( 1 ); // global object
46
47 int main()
48 {
49     cout << "    (global created before main)" << endl;
50
51     CreateAndDestroy second( 2 );    // local object
52     cout << "    (local automatic in main)" << endl;
53
54     static CreateAndDestroy third( 3 ); // local object
55     cout << "    (local static in main)" << endl;
56
57     create(); // call function to create objects
58
59     CreateAndDestroy fourth( 4 );    // local object
60     cout << "    (local automatic in main)" << endl;
61     return 0;
62 }
```



## Outline



### 3. Function definition

```
63
64 // Function to create objects
65 void create( void )
66 {
67     CreateAndDestroy fifth( 5 );
68     cout << "    (local automatic in create)" << endl;
69
70     static CreateAndDestroy sixth( 6 );
71     cout << "    (local static in create)" << endl;
72
73     CreateAndDestroy seventh( 7 );
74     cout << "    (local automatic in create)" << endl;
75 }
```

#### OUTPUT

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	

### Program Output

# 16.11 Using Data Members and Member Functions

- Classes provide **public** member functions
  - Set (i.e., write) or *get* (i.e., read) values of **private** data members
  - Adjustment of bank balance (a **private** data member of class **BankAccount**) by member function **computeInterest**
- Naming
  - Member function that *sets* **interestRate** typically named **setInterestRate**
  - Member function that *gets* **interestRate** would typically be called **getInterestRate**



# 16.11 Using Data Members and Member Functions (II)

- Do *set* and *get* capabilities effectively make data members **public**?
  - No!
  - Programmer decides what the function can set and what information the function can get
- **public** set functions should
  - Check attempts to modify data members
  - Ensure that the new value is appropriate for that data item
  - Example: an attempt to *set* the day of the month to 37 would be rejected
  - Programmer must include these features



## 16.12 A Subtle Trap: Returning a Reference to a Private Data Member

- Reference to an object
  - Alias for the name of the object
  - May be used on the left side of an assignment statement
  - Reference can receive a value, which changes the original object as well
- One way to use this capability (unfortunately!)
  - Have a **public** member function of a class return a non-**const** reference to a **private** data member
  - This reference can be modified, which changes the original data







## Outline



### 1. Define class

#### 1.1 Function prototypes

#### 1.2 badSetHour returns a reference

#### 1.3 Member variables

```
1 // Fig. 16.10: 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
```



## 1. Load header

### 1.1 Function definitions

```
23 // Fig. 16.10: time4.cpp
24 // Member function definitions for Time class.
25 #include "time4.h"
26
27 // Constructor function to initialize private data.
28 // Calls member function setTime to set variables.
29 // Default values are 0 (see class definition).
30 Time::Time( int hr, int min, int sec )
31     { setTime( hr, min, sec ); }
32
33 // Set the values of hour, minute, and second.
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 // Get the hour value
42 int Time::getHour() { return hour; }
43
44 // POOR PROGRAMMING PRACTICE:
45 // Returning a reference to a private data member.
46 int &Time::badSetHour( int hh )
47 {
48     hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
49
50     return hour; // DANGEROUS reference return
51 }
```



## 1.2 Declare reference

## 2. Change data using a reference

## 3. Print

```
52 // Fig. 16.10: fig16_10.cpp
53 // Demonstrating a public member function that
54 // returns a reference to a private data member.
55 // Time class has been trimmed for this example.
56 #include <iostream>
57
58 using std::cout;
59 using std::endl;
60
61 #include "time4.h"
62
63 int main()
64 {
65     Time t;
66     int &hourRef = t.badSetHour( 20 );
67
68     cout << "Hour before modification: " << hourRef;
69     hourRef = 30; // modification with invalid value
70     cout << "\nHour after modification: " << t.getHour();
71
72     // Dangerous: Function call that returns
73     // a reference can be used as an lvalue!
74     t.badSetHour(12) = 74;
75     cout << "\n\n*****\n"
76         << "POOR PROGRAMMING PRACTICE!!!!!!\n"
77         << "badSetHour as an lvalue, Hour: "
78         << t.getHour()
79         << "\n*****" << endl;
80
81     return 0;
82 }
```



Outline



**Program Output**

```
Hour before modification: 20
```

```
Hour after modification: 30
```

```
*****
```

```
POOR PROGRAMMING PRACTICE!!!!!!!
```

```
badSetHour as an lvalue, Hour: 74
```

```
*****
```

# 16.13 Assignment by Default Memberwise Copy

- Assignment operator (=)
  - Sets variables equal, i.e., ***x*** = ***y***;
  - Can be used to assign an object to another object of the same type
  - Memberwise copy — member by member copy  
**myObject1 = myObject2;**
- Objects may be
  - Passed as function arguments
  - Returned from functions (call-by-value default)
    - Use pointers for call by reference





## Outline



### 1. Define class

#### 1.1 Define member functions

```
1 // Fig. 16.11: fig16_11.cpp
2 // Demonstrating that class objects can be assigned
3 // to each other using default memberwise copy
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 // Simple Date class
10 class Date {
11 public:
12     Date( int = 1, int = 1, int = 1990 ); // default constructor
13     void print();
14 private:
15     int month;
16     int day;
17     int year;
18 };
19
20 // Simple Date constructor with no range checking
21 Date::Date( int m, int d, int y )
22 {
23     month = m;
24     day = d;
25     year = y;
26 }
27
28 // Print the Date in the form mm-dd-yyyy
29 void Date::print()
30 { cout << month << '-' << day << '-' << year; }
```



## Outline



### 2. Create Date objects

#### 2.1 Memberwise copy

### 3. Print values

```
31
32 int main()
33 {
34     Date date1( 7, 4, 1993 ), date2; // d2 defaults to 1/1/90
35
36     cout << "date1 = ";
37     date1.print();
38     cout << "\ndate2 = ";
39     date2.print();
40
41     date2 = date1; // assignment by default memberwise copy
42     cout << "\n\nAfter default memberwise copy, date2 = ";
43     date2.print();
44     cout << endl;
45
46     return 0;
47 }
```

date1 = 7-4-1993

date2 = 1-1-1990

After default memberwise copy, date2 = 7-4-1993

### Program Output

## 16.14 Software Reusability

- Object-oriented programmers
  - Concentrate on implementing useful classes
- Tremendous opportunity to capture and catalog classes
  - Accessed by large segments of the programming community
  - Class libraries exist for this purpose
- Software
  - Constructed from existing, well-defined, carefully tested, portable, widely available components
  - Speeds development of powerful, high-quality software

