Capitolo 6: Classes and Data Abstraction

<u>Outline</u>	
6.1	Introduction
6.2	Structure Definitions
6.3	Accessing Members of Structures
6.4	Implementing a User-Defined Type Time with a Struct
6.5	Implementing a Time Abstract Data Type with a Class
6.6	Class Scope and Accessing Class Members
6.7	Separating Interface from Implementation
6.8	Controlling Access to Members
6.9	Access Functions and Utility Functions
6.10	Initializing Class Objects: Constructors
6.11	Using Default Arguments with Constructors
6.12	Using Destructors
6.13	When Constructors and Destructors Are Called
6.14	Using Data Members and Member Functions
6.15	A Subtle Trap: Returning a Reference to a
	Private Data Member
6.16	Assignment by Default Memberwise Copy
6.17	Software Reusability



6.1 Introduction

- Object-oriented programming (OOP)
 - Encapsulates data (attributes) and functions (behavior) into packages called classes
- Information hiding
 - Implementation details are hidden within the classes themselves

Classes

- Classes are the standard unit of programming
- 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"

6.2 Structure Definitions

Structures

Aggregate data types built using elements of other types

```
struct Time {
    int hour;
    int minute;
    int second;
};
Structure tag
Structure members
```

- Members of the same structure must have unique names
- Two different structures may contain members of the same name
- Each structure definition must end with a semicolon

6.2 Structure Definitions

• Self-referential structure

- Contains a member that is a pointer to the same structure type
- Used for linked lists, queues, stacks and trees

struct

- Creates a new data type that is used to declare variables
- Structure variables are declared like variables of other types
- Example:



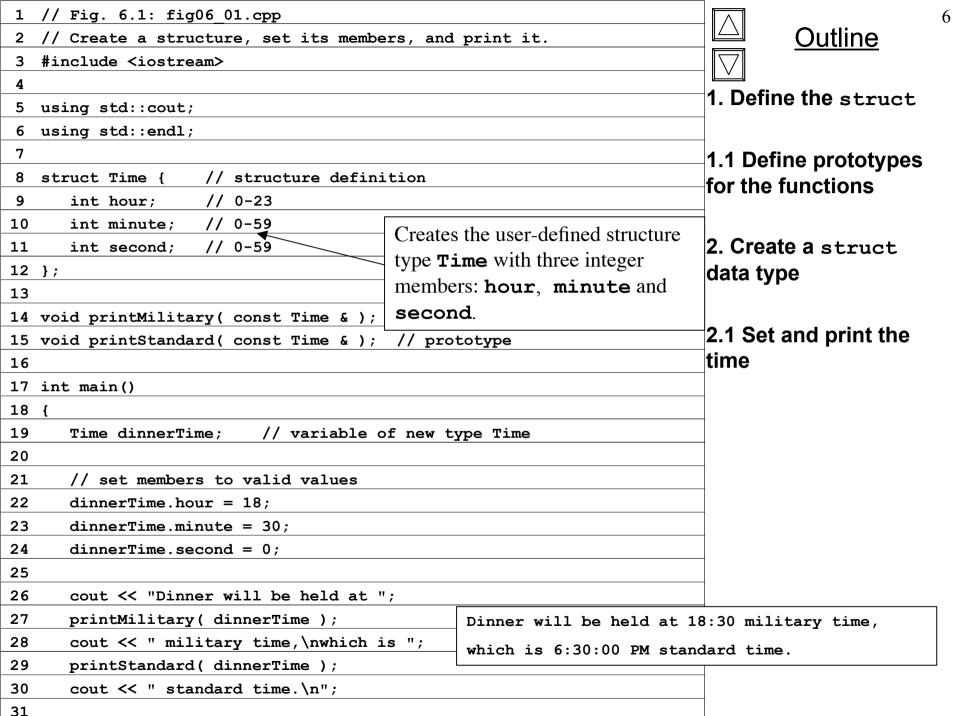
6.3 Accessing Members of Structures

- Member access operators:
 - Dot operator (.) for structures and objects
 - Arrow operator (->) for pointers
 - Print member **hour** of **timeObject**:

```
OR

timePtr = &timeObject;
cout << timePtr->hour;
```

- timePtr->hour is the same as (*timePtr).hour
- Parentheses required: * has lower precedence than .



```
32
      // set members to invalid values
33
      dinnerTime.hour = 29;
34
      dinnerTime.minute = 73;
35
      cout << "\nTime with invalid values: ";</pre>
36
37
      printMilitary( dinnerTime );
                                      Time with invalid values: 29:73
      cout << endl;</pre>
38
39
      return 0;
40 }
41
42 // Print the time in military format
43 void printMilitary( const Time &t )
44 {
45
      cout << ( t.hour < 10 ? "0" : "" ) << t.hour << ":"</pre>
46
           << ( t.minute < 10 ? "0" : "" ) << t.minute;
47 }
48
49 // Print the time in standard format
50 void printStandard( const Time &t )
51 {
52
      cout << ( ( t.hour == 0 || t.hour == 12 ) ?</pre>
53
                 12 : t.hour % 12 )
54
           << ":" << ( t.minute < 10 ? "0" : "" ) << t.minute
           << ":" << ( t.second < 10 ? "0" : "" ) << t.second
55
56
           << ( t.hour < 12 ? " AM" : " PM" );
57 }
```

Outline

2.2 Set the time to an invalid hour, then print it

3. Define the functions printMilitary and printStandard

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

Time with invalid values: 29:73

Outline

Program Output

Classes

- Model objects that have attributes (data members) and behaviors (member functions)
- Defined using keyword class
- Have a body delineated with braces ({ and })
- Class definitions terminate with a semicolon
- Example:

```
class Time {
                                              Public: and Private: are
  public:
                                              member-access specifiers.
      Time();
     void setTime( int, int, int );
                                             setTime, printMilitary, and
     void printMilitary();
      void printStandard();
                                             printStandard are member
  private:
                                             functions
                    // 0 - 23
      int hour;
                                             Time is the constructor.
                    // 0 - 59
      int minute;
                   // 0 - 59
10
      int second:
                                             hour, minute, and
11 };
                                             second are data members.
```



Member access specifiers

- Classes can limit the access to their member functions and data
- The three types of access a class can grant are:
 - **Public** Accessible wherever the program has access to an object of the class
 - **private** Accessible only to member functions of the class
 - **Protected** Similar to private and discussed later

Constructor

- Special member function that initializes the data members of a class object
- Cannot return values
- Have the same name as the class

- Class definition and declaration
 - Once a class has been defined, it can be used as a type in object, array and pointer declarations
 - Example:

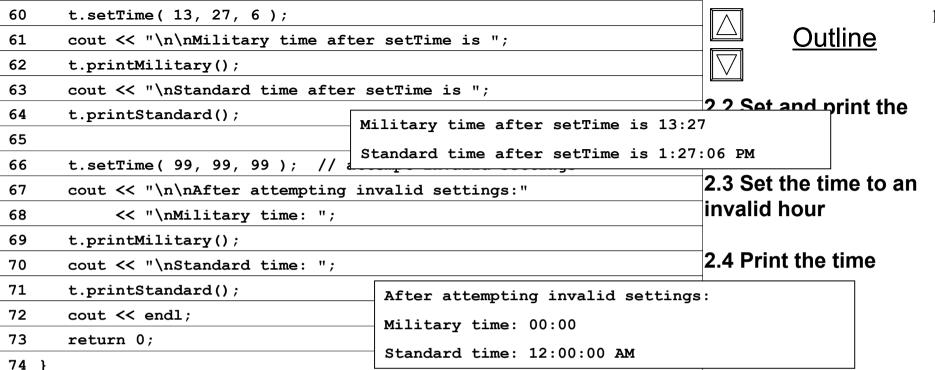
Note: The class name becomes the new type specifier.



```
1 // Fig. 6.3: fig06 03.cpp
                                                                                 Outline
2 // Time class.
3 #include <iostream>
4
                                                                        1. Define a Time class
5 using std::cout;
6 using std::endl;
                                                                        1.1 Define default
7
8 // Time abstract data type (ADT) definition
                                                                        values for the time
9 class Time {
10 public:
11
                                     // constructor
      Time();
      void setTime( int, int, int ); // set hour, minute, second
12
      void printMilitary();
                                     // print military time format
13
14
      void printStandard();
                                     // print standard time format
15 private:
      int hour;
                    // 0 - 23
16
      int minute; // 0 - 59
17
      int second; // 0 - 59
18
19 };
20
21 // Time constructor initializes each data member to zero.
22 // Ensures all Time objects start in a consistent state.
                                                                       Note the :: preceding
23 Time::Time() { hour = minute = second = 0; }
                                                                       the function names.
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 \ge 0 \&\& h < 24)? h : 0;
30
      minute = ( m \ge 0 \&\& m < 60 ) ? m : 0;
      second = (s \ge 0 \&\& s < 60)? s : 0;
31
32 }
```

33		1	
34 // Print Time in military format		<u> </u>	
35 void Time::printMilitary()			
36 {			
37 cout << (hour < 10 ? "0" : "") << hour <<	":"	1.2 Define the two	
38 << (minute < 10 ? "0" : "") << minute	;	functions	
39 }		printMilitary and	
40		printstandard	
41 // Print Time in standard format			
42 void Time::printStandard()		2. In main, create an	
43 {		object of class Time	
44 cout << ((hour == 0 hour == 12) ? 12 :	hour % 12)		
45 << ":" << (minute < 10 ? "0" : "") <<	45 << ":" << (minute < 10 ? "0" : "") << minute		
46 << ":" << (second < 10 ? "0" : "") <<	6 << ":" << (second < 10 ? "0" : "") << second		
47 << (hour < 12 ? " AM" : " PM");		(default) time	
48 }			
49			
50 // Driver to test simple class Time			
51 int main()			
52 {			
53 Time t; // instantiate object t of class Th	ne initial military tim	ne is 00:00	
54 Th	ne initial standard tim	ne is 12:00:00 AM	
55 cout << "The initial military time is ";			
56 t.printMilitary();	No	tice how functions are	
cout << "\nThe initia standard time is ";		led using the dot (.)	
t.printStandard();		erator.	
59			

 $^{\ @}$ 2000 Prentice Hall, Inc. All rights reserved.



```
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
```

Destructors

- Functions with the same name as the class but preceded with a tilde character (~)
- Cannot take arguments and cannot be overloaded
- Performs "termination housekeeping"
- Binary scope resolution operator (::)
 - Combines the class name with the member function name
 - Different classes can have member functions with the same name
- Format for defining member functions

```
ReturnType ClassName::MemberFunctionName(){
    ...
}
```

- If a 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

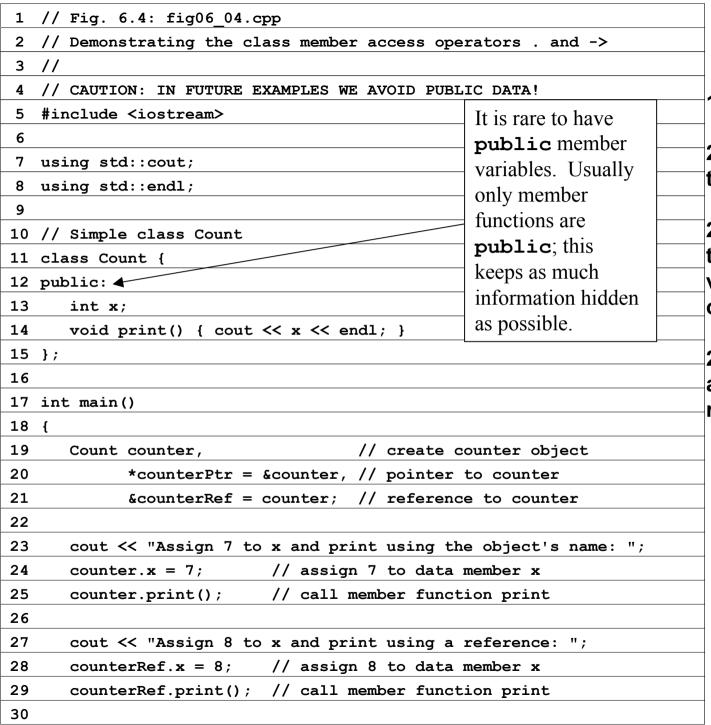


6.6 Class Scope and Accessing Class Members

- Class scope
 - Data members and member functions
- File scope
 - Nonmember functions
- Inside a scope
 - Members accessible by all member functions
 - Referenced by name
- Outside a scope
 - Members are referenced through handles
 - An object name, a reference to an object or a pointer to an object

6.6 Class Scope and Accessing Class Members

- Function scope
 - Variables only known to function they are defined in
 - Variables are destroyed after function completion
- 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



<u>Outline</u>



- 1. Class definition
- 2. Create an object of the class
- 2.1 Assign a value to the object. Print the value using the dot operator
- 2.2 Set a new value and print it using a reference

31	cout << "Assign 10 to x and print using a pointer: ";		20
32	counterPtr->x = 10; // assign 10 to data member x	Outline Outline	
33	<pre>counterPtr->print(); // call member function print</pre>		
34	return 0:	2.3 Set a new value and print it using a	
35 }		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

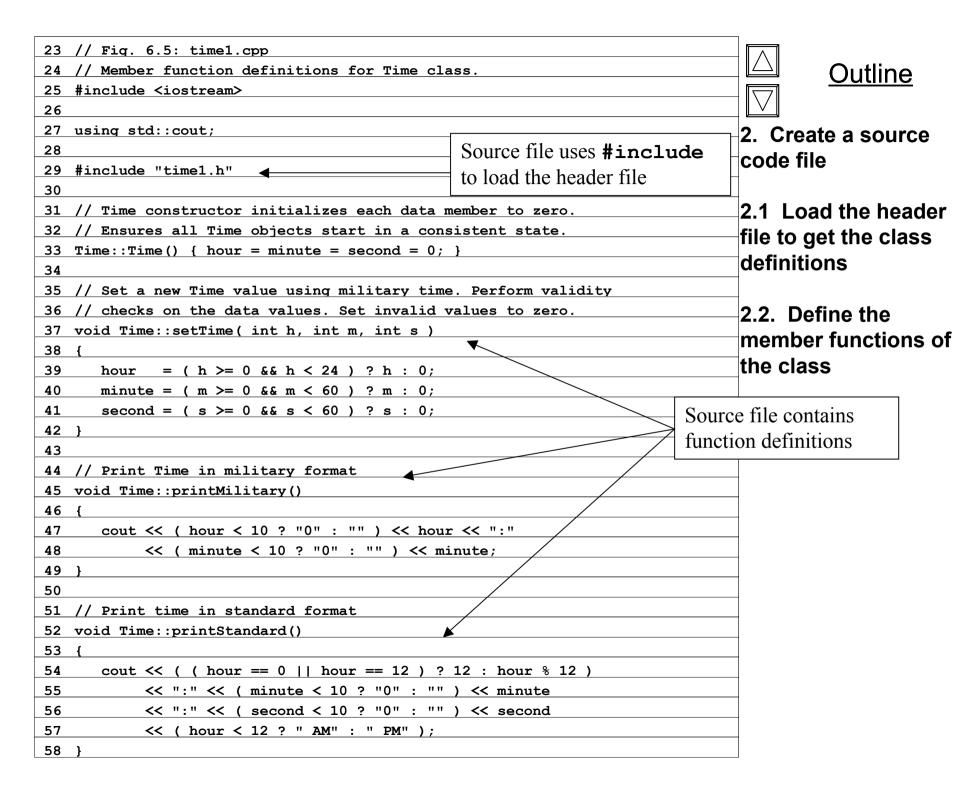
Program Output

6.7 Separating Interface from Implementation

- Separating interface from implementation
 - Makes it easier to modify programs
 - Header files
 - Contains class definitions and function prototypes
 - Source-code files
 - Contains member function definitions

1	// Fig. 6.5: time1.h					2
2	// Declaration of the Time class.					<u>Outline</u>
3	// Member functions are defined i	n time1.cpp				
4						ng the same
5	// prevent multiple inclusions of	header file			Time (class as before,
6	#ifndef TIME1_H		Do	ot (.) replaced with	undersc	fore (_) in file
7	#define TIME1_H		nar	me.	I	
8				If time 1 b (mr)	red 11)	is not defined
9	// Time abstract data type defini	tion		If time1.h (TIM (#ifndef) then i	— ′	
10	class Time {			TIME1 H). If TI		`
11	public:			defined, then every	_	•
12	Time();	// constructor		ignored.		
13	<pre>void setTime(int, int, int);</pre>	// set hour, m	inut	This prevents load	ing a he	ader file
14	<pre>void printMilitary();</pre>	// print milit	ary	multiple times.		
15	<pre>void printStandard();</pre>	// print stand	ard	time format		
16	private:					
17	int hour; // 0 - 23					
18	int minute; // 0 - 59					
19	int second; // 0 - 59					
20	};					
21						
22	#endif					

 $^{\ @}$ 2000 Prentice Hall, Inc. All rights reserved.



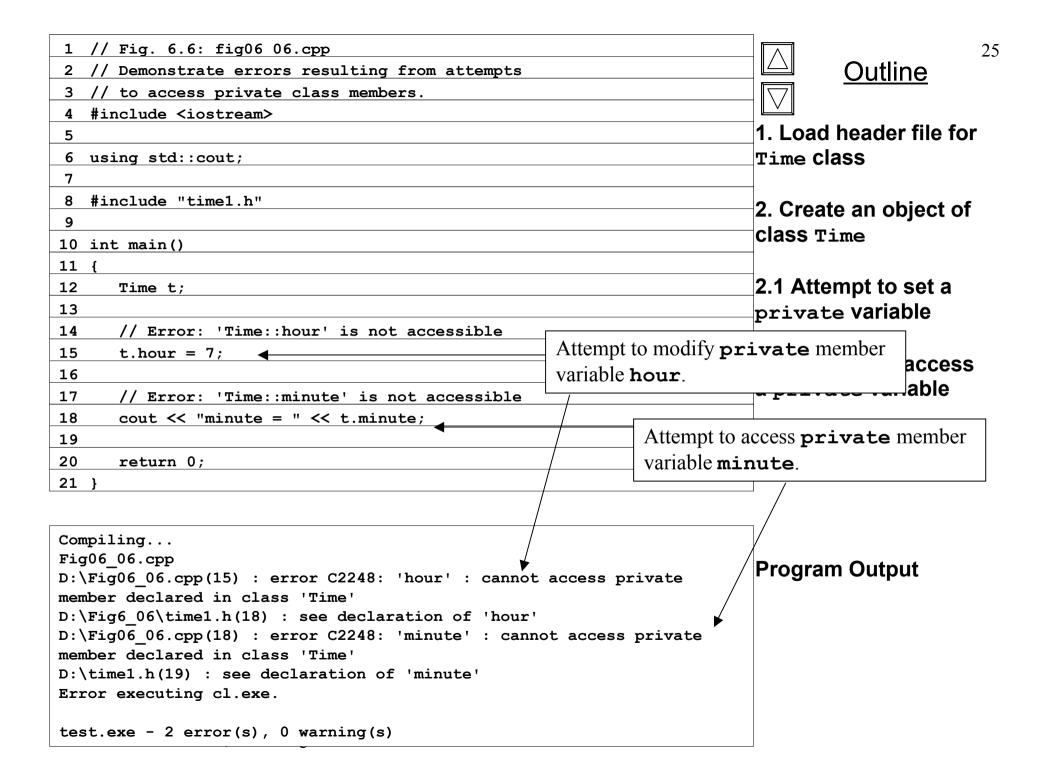
6.8 Controlling Access to Members

• public

- Presents clients with a view of the services the class provides (interface)
- Data and member functions are accessible

private

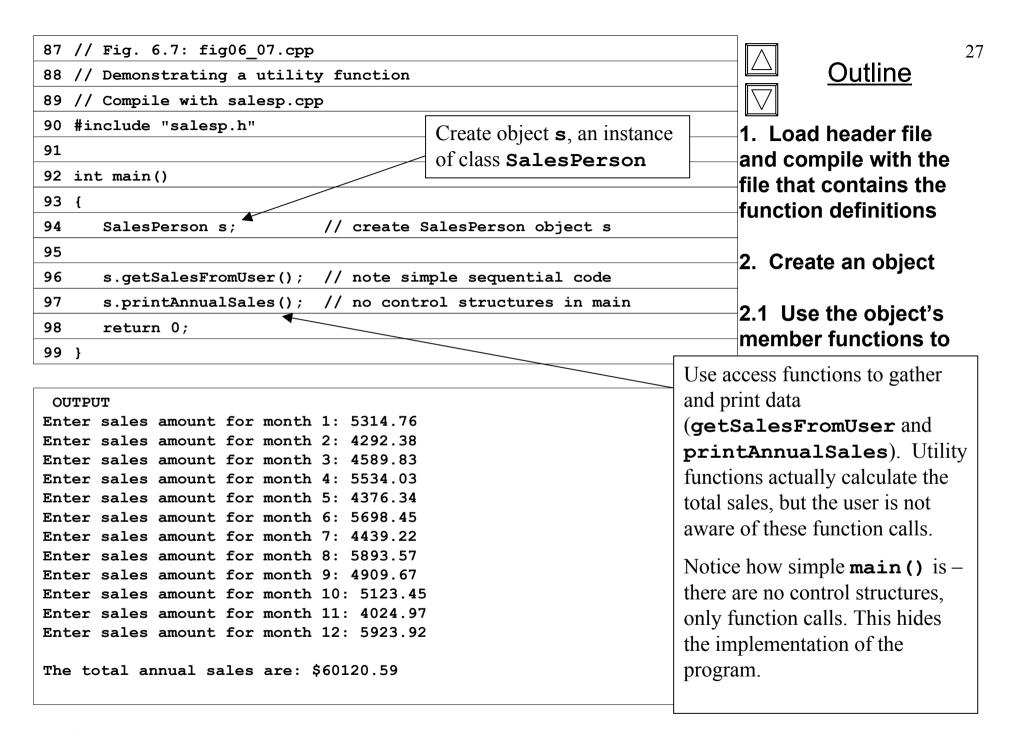
- Default access mode
- Data only accessible to member functions and friends
- **private** members only accessible through the **public** class interface using **public** member functions



6.9 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
 - Allow public functions to check private data
- Following example
 - Program to take in monthly sales and output the total
 - Implementation not shown, only access functions





6.10 Initializing Class Objects: Constructors

Constructors

- Initialize class members
- Same name as the class
- No return type
- Member variables can be initialized by the constructor or set afterwards

• Passing arguments to a constructor

- When an object of a class is declared, initializers can be provided
- Format of declaration with initializers:

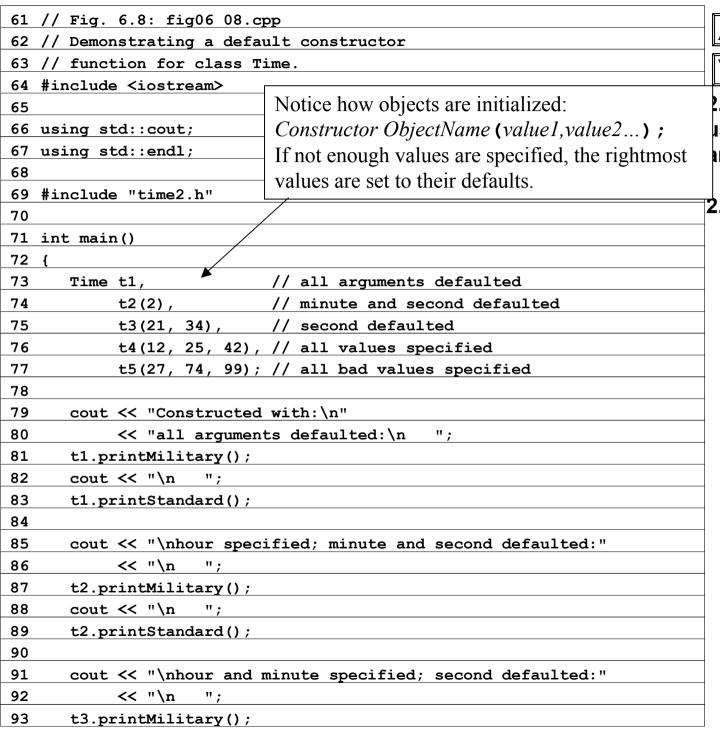
```
Class-type ObjectName( value1, value2, ...);
```

Default arguments may also be specified in the constructor prototype



```
1 // Fig. 6.8: time2.h
                                                                                           29
                                                                             Outline
2 // Declaration of the Time class.
3 // Member functions are defined in time2.cpp
                                                                    1. Define class Time
4
                                                                    and its default values
5 // preprocessor directives that
  // prevent multiple inclusions of header file
7 #ifndef TIME2 H
  #define TIME2 H
9
10 // Time abstract data type definition
11 class Time {
                                                                   Notice that default settings
12 public:
                                                                   for the three member
     13
                                                                   variables are set in
     void setTime( int, int, int ); // set hour, minute, second
14
                                                                   constructor prototype. No
     void printMilitary();
15
                                   // print military time format
                                                                   names are needed; the
16
     void printStandard();
                                   // print standard time format
                                                                   defaults are applied in the
17 private:
                                                                   order the member variables
                 // 0 - 23
18
     int hour;
                                                                   are declared.
     int minute; // 0 - 59
19
     int second; // 0 - 59
20
21 };
22
23 #endif
```

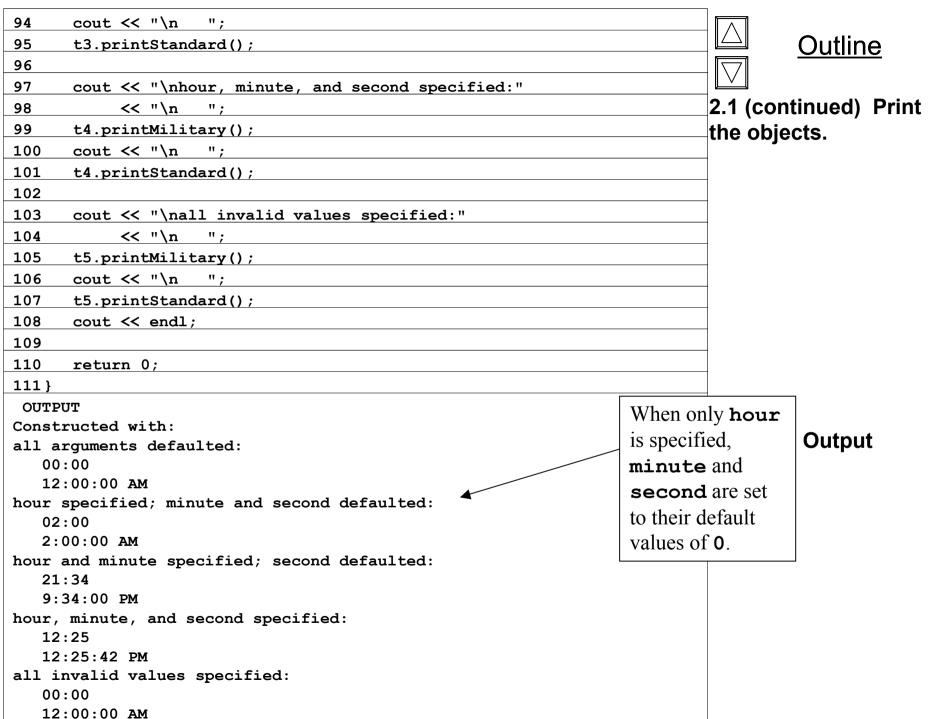
^{© 2000} Prentice Hall, Inc. All rights reserved.



<u>Outline</u>

2. Create objects using default arguments

2.1 Print the objects



6.12 Using Destructors

Destructors

- Are member function of class
- Perform termination housekeeping before the system reclaims the object's memory
- Complement of the constructor
- Name is tilde (~) followed by the class name (i.e., ~Time)
 - Recall that the constructor's name is the class name
- Receives no parameters, returns no value
- One destructor per class
 - No overloading allowed

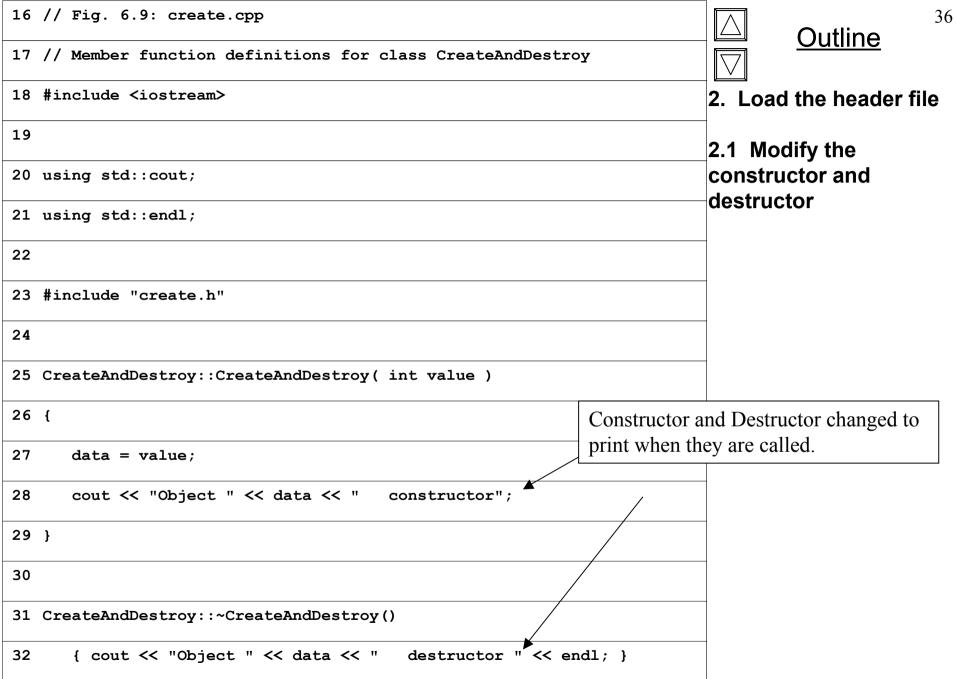
6.13 When Constructors and Destructors Are Called

- 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
- Automatic local objects
 - Constructors called when objects are defined
 - Destructors called when objects leave scope
 - i.e., when the block in which they are defined is exited
 - Destructors not called if the program ends with exit or abort

6.13 When Constructors and Destructors Are Called

- 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

1	// Fig. 6.9: create.h	35
2	// Definition of class CreateAndDestroy.	<u>Outline</u>
3	// Member functions defined in create.cpp.	
4	#ifndef CREATE_H	1. Create a header file
5	#define CREATE_H	
6		1.1 Include function
7	class CreateAndDestroy {	prototypes for the
8	public:	destructor and
9	CreateAndDestroy(int); // constructor	constructor
10	~CreateAndDestroy(); // destructor	
11	private:	
12	int data;	
13	};	
14		
15	#endif	



^{© 2000} Prentice Hall, Inc. All rights reserved.

33 // Fig. 6.9: fig06 09.cpp	
34 // Demonstrating the order in which constructors and	
35 // destructors are called.	
36 #include <iostream></iostream>	
37	3.
38 using std::cout;	0
39 using std::endl;	ty
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 }	

<u>Outline</u>

3. Create multiple objects of varying types

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

(global created before main)

(local automatic in create)

(local automatic in create)

(local automatic in main)

(local static in create)

(local automatic in main)

(local static in main)

Program Output

Notice how the order of the constructor and destructor call depends on the types of variables (automatic, global and static) they are associated with.

constructor

constructor

constructor

constructor

constructor

constructor

destructor

destructor

constructor

destructor

destructor

destructor

destructor

destructor

75 }

OUTPUT

Object 1

Object 2

Object 3

Object 5

Object 6

Object 7

Object 5

Object 4

Object 4

Object 2

Object 6

Object 3

Object 1

Object 7

6.14 Using Data Members and Member Functions

- Member functions
 - Allow clients of the class to set (i.e., write) or get (i.e., read) the values of private data members
 - Example:

Adjusting a customer's bank balance

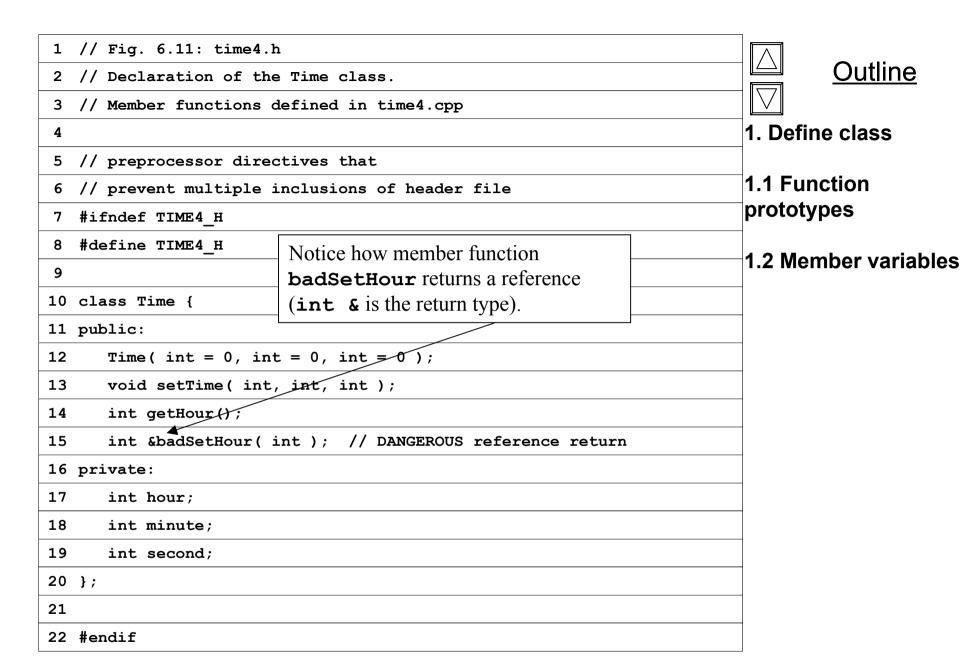
- private data member balance of a class BankAccount could be modified through the use of member function computeInterest
- A member function that sets data member interestRate could be called setInterestRate, and a member function that returns the interestRate could be called getInterestRate
- Providing set and get functions does not make private variables public
- A set function should ensure that the new value is valid



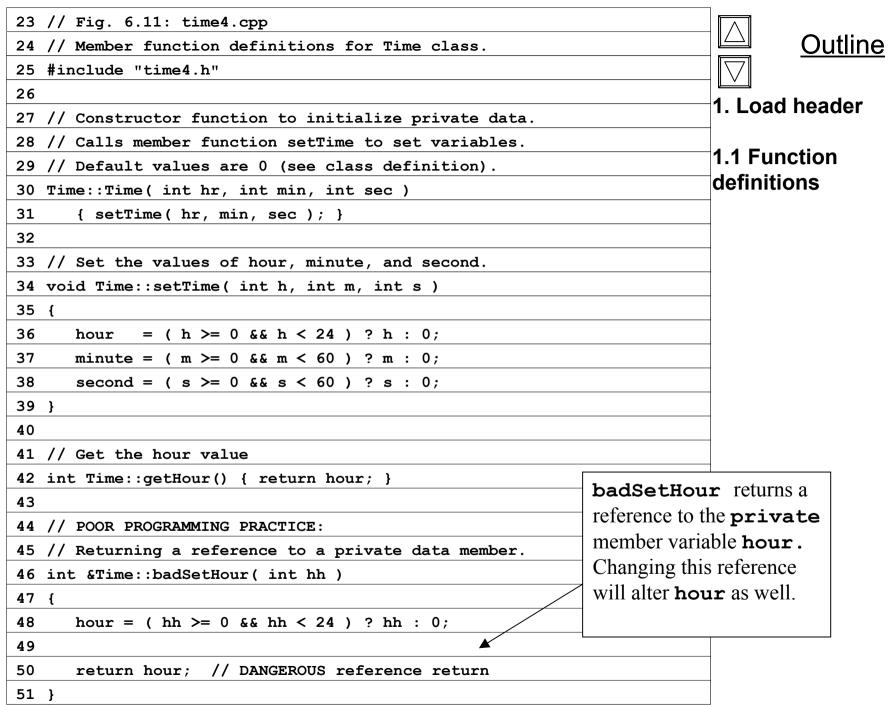
6.15 A Subtle Trap: Returning a Reference to a Private Data Member

- 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
- Returning references
 - public member functions can return non-const
 references to private data members
 - Should be avoided, breaks encapsulation

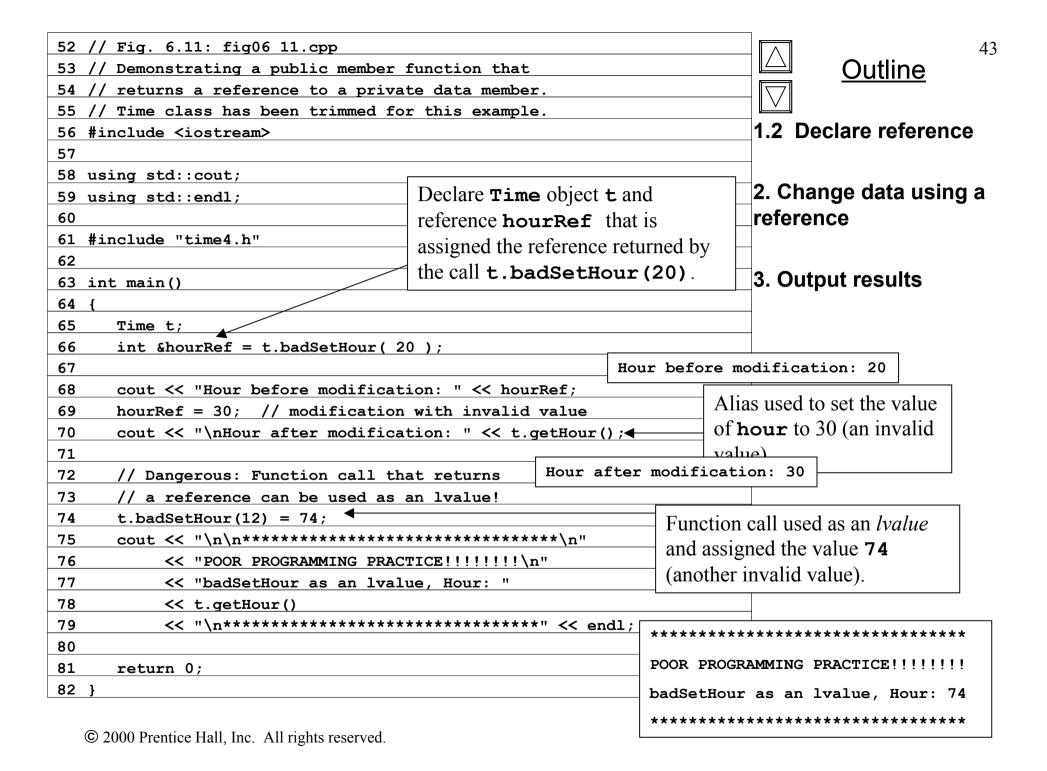




^{© 2000} Prentice Hall, Inc. All rights reserved.



^{© 2000} Prentice Hall, Inc. All rights reserved.





Outline

Program Output

HourRef used to change hour to an invalid value. Normally, the function setbadSetHour would not have allowed this. However, because it returned a reference, hour was changed directly.

6.16 Assignment by Default Memberwise Copy

- Assigning objects
 - An object can be assigned to another object of the same type using the assignment operator (=)
 - Member by member copy
- Objects may be
 - Passed as function arguments
 - Returned from functions (call-by-value default)



```
1 // Fig. 6.12: fig06 12.cpp
2 // Demonstrating that class objects can be assigned
3 // to each other using default memberwise copy
4 #include <iostream>
                                                              1. Define class
5
6 using std::cout;
7 using std::endl;
                                                              1.1 Define member
8
                                                              functions
9 // Simple Date class
10 class Date {
11 public:
12 Date( int = 1, int = 1, int = 1990 ); // default constructor
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; }
```

^{© 2000} Prentice Hall, Inc. All rights reserved.

```
31
32 int main()
33 {
34
      Date date1( 7, 4, 1993 ), date2; // d2 defaults to 1/1/90
                                                                            2. Create Date objects
35
36
      cout << "date1 = ";</pre>
                                                                            2.1 Memberwise copy
37
      date1.print();
      cout << "\ndate2 = ";</pre>
38
                                                                            3. Print values
39
      date2.print();
40
      date2 = date1; // assignment by default memberwise copy
41
      cout << "\n\nAfter default memberwise copy, date2 = ";</pre>
42
43
      date2.print();
                                       date2 set equal to date1,
44
      cout << endl;</pre>
                                       and all member variables
45
                                       are copied.
46
      return 0;
47 }
date1 = 7-4-1993
                                                                           Program Output
date2 = 1-1-1990
```

© 2000 Prentice Hall, Inc. All rights reserved.

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

6.17 Software Reusability

- Software resusability
 - Implementation of useful classes
 - Class libraries exist to promote reusability
 - Allows for construction of programs from existing, well-defined, carefully tested, well-documented, portable, widely available components
 - Speeds development of powerful, high-quality software

