

Note: *C How to Program*, Chapter 18 is a copy of *C++ How to Program* Chapter 10. We have not renumbered the PowerPoint Slides.

Chapter 10 Classes: A Deeper Look, Part 2

C++ How to Program, 8/e

©1992–2012 by Pearson Education, Inc. All Rights Reserved.

10.2 `const` (Constant) Objects and `const` Member Functions

- ▶ You may use keyword `const` to specify that an object is not modifiable and that any attempt to modify the object should result in a compilation error.
- ▶ C++ disallows member function calls for `const` objects unless the member functions themselves are also declared `const`.
 - True even for *get member functions that do not modify the object*.
- ▶ A member function is specified as `const` *both in its prototype and in its definition*.

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

1 // Fig. 10.1: Time.h
2 // Time class definition with const member functions.
3 // Member functions defined in Time.cpp.
4 #ifndef TIME_H
5 #define TIME_H
6
7 class Time
8 {
9 public:
10    Time( int = 0, int = 0, int = 0 ); // default constructor
11
12    // set functions
13    void setTime( int, int, int ); // set time
14    void setHour( int ); // set hour
15    void setMinute( int ); // set minute
16    void setSecond( int ); // set second
17

```

Fig. 10.1 | Time class definition with const member functions. (Part 1 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.


```

18    // get functions (normally declared const)
19    int getHour() const; // return hour
20    int getMinute() const; // return minute
21    int getSecond() const; // return second
22
23    // print functions (normally declared const)
24    void printUniversal() const; // print universal time
25    void printStandard(); // print standard time (should be const)
26 private:
27    int hour; // 0 - 23 (24-hour clock format)
28    int minute; // 0 - 59
29    int second; // 0 - 59
30 }; // end class Time
31
32 #endif

```

Fig. 10.1 | Time class definition with const member functions. (Part 2 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.




```

1 // Fig. 10.2: Time.cpp
2 // Time class member-function definitions.
3 #include <iostream>
4 #include <iomanip>
5 #include <stdexcept>
6 #include "Time.h" // include definition of class Time
7 using namespace std;
8
9 // constructor function to initialize private data;
10 // calls member function setTime to set variables;
11 // default values are 0 (see class definition)
12 Time::Time( int hour, int minute, int second )
13 {
14     setTime( hour, minute, second );
15 } // end Time constructor
16
17 // set hour, minute and second values
18 void Time::setTime( int hour, int minute, int second )
19 {
20     setHour( hour );
21     setMinute( minute );
22     setSecond( second );
23 } // end function setTime
24

```

Fig. 10.2 | Time class member-function definitions. (Part 1 of 5.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.




```

25 // set hour value
26 void Time::setHour( int h )
27 {
28     if ( h >= 0 && h < 24 )
29         hour = h;
30     else
31         throw invalid_argument( "hour must be 0-23" );
32 } // end function setHour
33
34 // set minute value
35 void Time::setMinute( int m )
36 {
37     if ( m >= 0 && m < 60 )
38         minute = m;
39     else
40         throw invalid_argument( "minute must be 0-59" );
41 } // end function setMinute
42

```

Fig. 10.2 | Time class member-function definitions. (Part 2 of 5.)


©1992–2012 by Pearson Education, Inc.
All Rights Reserved.



```
43 // set second value
44 void Time::setSecond( int s )
45 {
46     if ( s >= 0 && s < 60 )
47         second = s;
48     else
49         throw invalid_argument( "second must be 0-59" );
50 } // end function setSecond
51
52 // return hour value
53 int Time::getHour() const // get functions should be const
54 {
55     return hour;
56 } // end function getHour
57
58 // return minute value
59 int Time::getMinute() const
60 {
61     return minute;
62 } // end function getMinute
63
```

Fig. 10.2 | Time class member-function definitions. (Part 3 of 5.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.



```
64 // return second value
65 int Time::getSecond() const
66 {
67     return second;
68 } // end function getSecond
69
```

Fig. 10.2 | Time class member-function definitions. (Part 4 of 5.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

70 // print Time in universal-time format (HH:MM:SS)
71 void Time::printUniversal() const
72 {
73     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
74         << setw( 2 ) << minute << ":" << setw( 2 ) << second;
75 } // end function printUniversal
76
77 // print Time in standard-time format (HH:MM:SS AM or PM)
78 void Time::printStandard() // note lack of const declaration
79 {
80     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
81         << ":" << setfill( '0' ) << setw( 2 ) << minute
82         << ":" << setw( 2 ) << second << ( hour < 12 ? " AM" : " PM" );
83 } // end function printStandard

```

Fig. 10.2 | Time class member-function definitions. (Part 5 of 5.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

1 // Fig. 10.3: fig10_03.cpp
2 // Attempting to access a const object with non-const member functions.
3 #include "Time.h" // include Time class definition
4
5 int main()
6 {
7     Time wakeUp( 6, 45, 0 ); // non-constant object
8     const Time noon( 12, 0, 0 ); // constant object
9
10
11     wakeUp.setHour( 18 ); // OBJECT      MEMBER FUNCTION
12                          // non-const  non-const
13     noon.setHour( 12 ); // const      non-const
14
15     wakeUp.getHour(); // non-const  const
16
17     noon.getMinute(); // const      const
18     noon.printUniversal(); // const    const
19
20     noon.printStandard(); // const    non-const
21 } // end main

```

Fig. 10.3 | const objects and const member functions. (Part 1 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

10.3 Composition: Objects as Members of Classes

- ▶ **Composition**
 - Sometimes referred to as a **has-a relationship**
 - A class can have objects of other classes as members
- ▶ An object's constructor can pass arguments to member-object constructors via member initializers.

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.


```

1 // Fig. 10.8: Date.h
2 // Date class definition; Member functions defined in Date.cpp
3 #ifndef DATE_H
4 #define DATE_H
5
6 class Date
7 {
8 public:
9     static const int monthsPerYear = 12; // number of months in a year
10    Date( int = 1, int = 1, int = 1900 ); // default constructor
11    void print() const; // print date in month/day/year format
12    ~Date(); // provided to confirm destruction order
13 private:
14    int month; // 1-12 (January-December)
15    int day; // 1-31 based on month
16    int year; // any year
17
18    // utility function to check if day is proper for month and year
19    int checkDay( int ) const;
20 }; // end class Date
21
22 #endif

```

Fig. 10.8 | Date class definition.

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.




```

1 // Fig. 10.9: Date.cpp
2 // Date class member-function definitions.
3 #include <iostream>
4 #include <stdexcept>
5 #include "Date.h" // include Date class definition
6 using namespace std;
7
8 // constructor confirms proper value for month; calls
9 // utility function checkDay to confirm proper value for day
10 Date::Date( int mn, int dy, int yr )
11 {
12     if ( mn > 0 && mn <= monthsPerYear ) // validate the month
13         month = mn;
14     else
15         throw invalid_argument( "month must be 1-12" );
16
17     year = yr; // could validate yr
18     day = checkDay( dy ); // validate the day
19
20     // output Date object to show when its constructor is called
21     cout << "Date object constructor for date ";
22     print();
23     cout << endl;
24 } // end Date constructor

```

Fig. 10.9 | Date class member-function definitions. (Part 1 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.



```

25
26 // print Date object in form month/day/year
27 void Date::print() const
28 {
29     cout << month << '/' << day << '/' << year;
30 } // end function print
31
32 // output Date object to show when its destructor is called
33 Date::~Date()
34 {
35     cout << "Date object destructor for date ";
36     print();
37     cout << endl;
38 } // end ~Date destructor
39

```

Fig. 10.9 | Date class member-function definitions. (Part 2 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

40 // utility function to confirm proper day value based on
41 // month and year; handles leap years, too
42 int Date::checkDay( int testDay ) const
43 {
44     static const int daysPerMonth[ monthsPerYear + 1 ] =
45         { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
46
47     // determine whether testDay is valid for specified month
48     if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
49         return testDay;
50
51     // February 29 check for leap year
52     if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
53         ( year % 4 == 0 && year % 100 != 0 ) ) )
54         return testDay;
55
56     throw invalid_argument( "Invalid day for current month and year" );
57 } // end function checkDay

```

Fig. 10.9 | Date class member-function definitions. (Part 3 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

1 // Fig. 10.10: Employee.h
2 // Employee class definition showing composition.
3 // Member functions defined in Employee.cpp.
4 #ifndef EMPLOYEE_H
5 #define EMPLOYEE_H
6
7 #include <string>
8 #include "Date.h" // include Date class definition
9 using namespace std;
10

```

Fig. 10.10 | Employee class definition showing composition. (Part 1 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.


```

11 class Employee
12 {
13 public:
14     Employee( const string &, const string &,
15             const Date &, const Date & );
16     void print() const;
17     ~Employee(); // provided to confirm destruction order
18 private:
19     string firstName; // composition: member object
20     string lastName; // composition: member object
21     const Date birthDate; // composition: member object
22     const Date hireDate; // composition: member object
23 }; // end class Employee
24
25 #endif

```

Fig. 10.10 | Employee class definition showing composition. (Part 2 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

1 // Fig. 10.11: Employee.cpp
2 // Employee class member-function definitions.
3 #include <iostream>
4 #include "Employee.h" // Employee class definition
5 #include "Date.h" // Date class definition
6 using namespace std;
7
8 // constructor uses member initializer list to pass initializer
9 // values to constructors of member objects
10 Employee::Employee( const string &first, const string &last,
11                   const Date &dateOfBirth, const Date &dateOfHire )
12     : firstName( first ), // initialize firstName
13       lastName( last ), // initialize lastName
14       birthDate( dateOfBirth ), // initialize birthDate
15       hireDate( dateOfHire ) // initialize hireDate
16 {
17     // output Employee object to show when constructor is called
18     cout << "Employee object constructor: "
19          << firstName << ' ' << lastName << endl;
20 } // end Employee constructor
21

```

Fig. 10.11 | Employee class member-function definitions, including constructor with a member initializer list. (Part 1 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

22 // print Employee object
23 void Employee::print() const
24 {
25     cout << lastName << ", " << firstName << " Hired: ";
26     hireDate.print();
27     cout << " Birthday: ";
28     birthDate.print();
29     cout << endl;
30 } // end function print
31
32 // output Employee object to show when its destructor is called
33 Employee::~Employee()
34 {
35     cout << "Employee object destructor: "
36         << lastName << ", " << firstName << endl;
37 } // end ~Employee destructor

```

Fig. 10.11 | Employee class member-function definitions, including constructor with a member initializer list. (Part 2 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

1 // Fig. 10.12: fig10_12.cpp
2 // Demonstrating composition--an object with member objects.
3 #include <iostream>
4 #include "Employee.h" // Employee class definition
5 using namespace std;
6
7 int main()
8 {
9     Date birth( 7, 24, 1949 );
10    Date hire( 3, 12, 1988 );
11    Employee manager( "Bob", "Blue", birth, hire );
12
13    cout << endl;
14    manager.print();
15 } // end main

```

Fig. 10.12 | Demonstrating composition—an object with member objects. (Part 1 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

◀ ▶

```

Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Blue

Blue, Bob Hired: 3/12/1988 Birthday: 7/24/1949
Employee object destructor: Blue, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949

```

There are actually five constructor calls when an `Employee` is constructed—two calls to the `string` class's constructor (lines 12–13 of Fig. 10.11), two calls to the `Date` class's default copy constructor (lines 14–15 of

Fig. 10.12 | Demonstrating composition—an object with member objects. (Part 2 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

◀ ▶

10.5 Using the `this` Pointer

- ▶ How do member functions know *which* object's data members to manipulate? Every object has access to its own address through a pointer called `this` (a C++ keyword).
- ▶ The `this` pointer is not part of the object itself.
 - The `this` pointer is passed (by the compiler) as an implicit argument to each of the object's non-`static` member functions.
- ▶ Objects use the `this` pointer implicitly or explicitly to reference their data members and member functions.
- ▶ The type of the `this` pointer depends on the type of the object and whether the member function in which `this` is used is declared `const`.

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

1 // Fig. 10.14: fig10_14.cpp
2 // Using the this pointer to refer to object members.
3 #include <iostream>
4 using namespace std;
5
6 class Test
7 {
8 public:
9     Test( int = 0 ); // default constructor
10    void print() const;
11 private:
12    int x;
13 }; // end class Test
14
15 // constructor
16 Test::Test( int value )
17     : x( value ) // initialize x to value
18 {
19     // empty body
20 } // end constructor Test
21

```

Fig. 10.14 | using the this pointer to refer to object members. (Part 1 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

22 // print x using implicit and explicit this pointers;
23 // the parentheses around *this are required
24 void Test::print() const
25 {
26     // implicitly use the this pointer to access the member x
27     cout << "    x = " << x;
28
29     // explicitly use the this pointer and the arrow operator
30     // to access the member x
31     cout << "\n this->x = " << this->x;
32
33     // explicitly use the dereferenced this pointer and
34     // the dot operator to access the member x
35     cout << "\n(*this).x = " << (*this).x << endl;
36 } // end function print
37
38 int main()
39 {
40     Test testObject( 12 ); // instantiate and initialize testObject
41
42     testObject.print();
43 } // end main

```

Fig. 10.14 | using the this pointer to refer to object members. (Part 2 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```
x = 12  
this->x = 12  
(*this).x = 12
```

Fig. 10.14 | using the `this` pointer to refer to object members. (Part 3 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

10.5 Using the `this` Pointer (cont.)

- ▶ Another use of the `this` pointer is to enable **cascaded member-function calls**
 - invoking multiple functions in the same statement
- ▶ The program of Figs. 10.15–10.17 modifies class `Time`'s *set functions* `setTime`, `setHour`, `setMinute` and `setSecond` such that each returns a reference to a `Time` object to enable cascaded member-function calls.

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

1 // Fig. 10.15: Time.h
2 // Cascading member function calls.
3
4 // Time class definition.
5 // Member functions defined in Time.cpp.
6 #ifndef TIME_H
7 #define TIME_H
8
9 class Time
10 {
11 public:
12     Time( int = 0, int = 0, int = 0 ); // default constructor
13
14     // set functions (the Time & return types enable cascading)
15     Time &setTime( int, int, int ); // set hour, minute, second
16     Time &setHour( int ); // set hour
17     Time &setMinute( int ); // set minute
18     Time &setSecond( int ); // set second
19

```

Fig. 10.15 | Time class modified to enable cascaded member-function calls. (Part 1 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

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

```

Fig. 10.15 | Time class modified to enable cascaded member-function calls. (Part 2 of 2.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

1 // Fig. 10.16: Time.cpp
2 // Time class member-function definitions.
3 #include <iostream>
4 #include <iomanip>
5 #include "Time.h" // Time class definition
6 using namespace std;
7
8 // constructor function to initialize private data;
9 // calls member function setTime to set variables;
10 // default values are 0 (see class definition)
11 Time::Time( int hr, int min, int sec )
12 {
13     setTime( hr, min, sec );
14 } // end Time constructor
15
16 // set values of hour, minute, and second
17 Time &Time::setTime( int h, int m, int s ) // note Time & return
18 {
19     setHour( h );
20     setMinute( m );
21     setSecond( s );
22     return *this; // enables cascading
23 } // end function setTime

```

Fig. 10.16 | Time class member-function definitions modified to enable cascaded member-function calls. (Part 1 of 5.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

24
25 // set hour value
26 Time &Time::setHour( int h ) // note Time & return
27 {
28     if ( h >= 0 && h < 24 )
29         hour = h;
30     else
31         throw invalid_argument( "hour must be 0-23" );
32
33     return *this; // enables cascading
34 } // end function setHour
35
36 // set minute value
37 Time &Time::setMinute( int m ) // note Time & return
38 {
39     if ( m >= 0 && m < 60 )
40         minute = m;
41     else
42         throw invalid_argument( "minute must be 0-59" );
43
44     return *this; // enables cascading
45 } // end function setMinute
46

```

Fig. 10.16 | Time class member-function definitions modified to enable cascaded member-function calls. (Part 2 of 5.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```

47 // set second value
48 Time &Time::setSecond( int s ) // note Time & return
49 {
50     if ( s >= 0 && s < 60 )
51         second = s;
52     else
53         throw invalid_argument( "second must be 0-59" );
54
55     return *this; // enables cascading
56 } // end function setSecond
57
58 // get hour value
59 int Time::getHour() const
60 {
61     return hour;
62 } // end function getHour
63

```

Fig. 10.16 | Time class member-function definitions modified to enable cascaded member-function calls. (Part 3 of 5.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.


```

64 // get minute value
65 int Time::getMinute() const
66 {
67     return minute;
68 } // end function getMinute
69
70 // get second value
71 int Time::getSecond() const
72 {
73     return second;
74 } // end function getSecond
75
76 // print Time in universal-time format (HH:MM:SS)
77 void Time::printUniversal() const
78 {
79     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
80         << setw( 2 ) << minute << ":" << setw( 2 ) << second;
81 } // end function printUniversal
82

```

Fig. 10.16 | Time class member-function definitions modified to enable cascaded member-function calls. (Part 4 of 5.)


©1992–2012 by Pearson Education, Inc.
All Rights Reserved.



```
83 // print Time in standard-time format (HH:MM:SS AM or PM)
84 void Time::printStandard() const
85 {
86     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
87         << ":" << setfill( '0' ) << setw( 2 ) << minute
88         << ":" << setw( 2 ) << second << ( hour < 12 ? " AM" : " PM" );
89 } // end function printStandard
```

Fig. 10.16 | Time class member-function definitions modified to enable cascaded member-function calls. (Part 5 of 5.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.



```
1 // Fig. 10.17: fig10_17.cpp
2 // Cascading member-function calls with the this pointer.
3 #include <iostream>
4 #include "Time.h" // Time class definition
5 using namespace std;
6
```

Fig. 10.17 | Cascading member-function calls with the this pointer. (Part 1 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```
7 int main()
8 {
9     Time t; // create Time object
10
11     // cascaded function calls
12     t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );
13
14     // output time in universal and standard formats
15     cout << "Universal time: ";
16     t.printUniversal();
17
18     cout << "\nStandard time: ";
19     t.printStandard();
20
21     cout << "\n\nNew standard time: ";
22
23     // cascaded function calls
24     t.setTime( 20, 20, 20 ).printStandard();
25     cout << endl;
26 } // end main
```

Fig. 10.17 | Cascading member-function calls with the `this` pointer.
(Part 2 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.

```
Universal time: 18:30:22
Standard time: 6:30:22 PM

New standard time: 8:20:20 PM
```

Fig. 10.17 | Cascading member-function calls with the `this` pointer.
(Part 3 of 3.)

©1992–2012 by Pearson Education, Inc.
All Rights Reserved.