

Lesson 3 - Outline

- **Assignment by Default Memberwise Copy**
- **“const” Objects**
- **“const” Member Functions**
- **The Constructor Initializer**
- **Composition: Objects as Members of Classes**



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. Define class

1.1 Define member functions

```
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>
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; }
```



```
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 }
```

2. Create Date objects

2.1 Memberwise copy

3. Print values

date2 set equal to date1,
and all member variables
are copied.

Program Output

```
date1 = 7-4-1993
date2 = 1-1-1990
```

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



“const” Objects

- Principle of least privilege
 - Only give objects permissions they need, no more
- Keyword **const**
 - Specify that an object is not modifiable
 - Any attempt to modify the object is a syntax error
 - Example

```
const Time noon( 12, 0, 0 );
```

- Declares a **const** object **noon** of class **Time**
- Initializes this object to value **12**



“const” Member Functions

- **const** objects require **const** functions
 - Member functions declared **const** cannot modify their object
 - **const** must be specified in function prototype and definition
 - Prototype:

```
ReturnType FuncName(param1,...) const;
```
 - Definition:

```
ReturnType FuncName(param1,...) const { ... }
```
 - Example:

```
int A::getValue() const  
{  
    return privateDataMember;  
}
```
- Constructors (and Destructors) cannot be **const**
 - They need to initialize variables, therefore modifying them





Outline



1. Class definition

1.1 Function prototypes

1.2 Member variables

```

1 // Fig. 7.1: time5.h
2 // Declaration of the class Time.
3 // Member functions defined in time5.cpp
4 #ifndef TIME5_H
5 #define TIME5_H
6
7 class Time {
8 public:
9     Time( int = 0, int = 0, int = 0 ); // default constructor
10
11     // set functions
12     void setTime( int, int, int ); // set time
13     void setHour( int ); // set hour
14     void setMinute( int ); // set minute
15     void setSecond( int ); // set second
16
17     // get functions (normally declared const)
18     int getHour() const; // return hour
19     int getMinute() const; // return minute
20     int getSecond() const; // return second
21
22     // print functions (normally declared const)
23     void printMilitary() const; // print military time
24     void printStandard(); // print standard time
25 private:
26     int hour; // 0 - 23
27     int minute; // 0 - 59
28     int second; // 0 - 59
29 };
30
31 #endif

```

const
functions

non-const
functions



Source Code

1. Load Header

1.1 Function definitions

The constructor is non-**const** but it can be called for **const** objects.

```
32 // Fig. 7.1: time5.cpp
33 // Member function definitions for Time class.
34 #include <iostream>
35
36 using std::cout;
37
38 #include "time5.h"
39
40 // Constructor function to initialize private data.
41 // Default values are 0 (see class definition).
42 Time::Time( int hr, int min, int sec )
43     { setTime( hr, min, sec ); }
44
45 // Set the values of hour, minute, and second.
46 void Time::setTime( int h, int m, int s )
47 {
48     setHour( h );
49     setMinute( m );
50     setSecond( s );
51 }
52
53 // Set the hour value
54 void Time::setHour( int h )
55     { hour = ( h >= 0 && h < 24 ) ? h : 0; }
56
57 // Set the minute value
58 void Time::setMinute( int m )
59     { minute = ( m >= 0 && m < 60 ) ? m : 0; }
60
61 // Set the second value
62 void Time::setSecond( int s )
63     { second = ( s >= 0 && s < 60 ) ? s : 0; }
```




Keyword **const** in function definition and prototype.

1.1 Function definitions

1.2 Purposely leave out **const** keyword for **printStandard**

Non-**const** functions cannot use **const** objects, even if they don't modify them (such as **printStandard**).

```
64
65 // Get the hour value
66 int Time::getHour() const { return hour; }
67
68 // Get the minute value
69 int Time::getMinute() const { return minute; }
70
71 // Get the second value
72 int Time::getSecond() const { return second; }
73
74 // Display military format time: HH:MM
75 void Time::printMilitary() const
76 {
77     cout << ( hour < 10 ? "0" : "" ) << hour << ":"
78         << ( minute < 10 ? "0" : "" ) << minute;
79 }
80
81 // Display standard format time: HH:MM:SS AM (or PM)
82 void Time::printStandard() // should be const
83 {
84     cout << ( ( hour == 12 ) ? 12 : hour % 12 ) << ":"
85         << ( minute < 10 ? "0" : "" ) << minute << ":"
86         << ( second < 10 ? "0" : "" ) << second
87         << ( hour < 12 ? " AM" : " PM" );
88 }
```

Outline

1. Initialize variables

2. Attempt to use non-const functions with const objects

Program Output

Compiler errors generated.

```

89 // Fig. 7.1: fig07_01.cpp
90 // Attempting to access a const object with
91 // non-const member functions.
92 #include "time5.h"
93
94 int main()
95 {
96     Time wakeUp( 6, 45, 0 );           // non-constant object
97     const Time noon( 12, 0, 0 );       // constant object
98
99                                     // MEMBER FUNCTION    OBJECT
100    wakeUp.setHour( 18 );               // non-const        non-const
101
102    noon.setHour( 12 );                 // non-const        const
103
104    wakeUp.getHour();                   // const            non-const
105
106    noon.getMinute();                   // const            const
107    noon.printMilitary();                // const            const
108    noon.printStandard();                // non-const        const
109    return 0;
110 }

```

```

Compiling...
Fig07_01.cpp
d:fig07_01.cpp(14) : error C2662: 'setHour' : cannot convert 'this'
pointer from 'const class Time' to 'class Time &'
Conversion loses qualifiers
d:\fig07_01.cpp(20) : error C2662: 'printStandard' : cannot convert
'this' pointer from 'const class Time' to 'class Time &'
Conversion loses qualifiers
Time5.cpp
Error executing cl.exe.

```

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

The Constructor Initializer

- Unlike other functions, a constructor may also contain a *constructor initializer list*:

- Example:

```
Time::Time( int hr, int min, int sec )  
    : h(hr), m(min), s(sec) { ... }
```

– Where, for instance, `: h(hr)` initializes `h` to `hr`

- Suitable for initializing all *data members*
- **const** *data members*, and *references* **must** be initialized using the *constructor initializer list*



Composition: Objects as Members of Classes

- A class has objects of other classes as members
- Construction of objects
 - Member objects constructed in order declared
 - Not in order of constructor's member initializer list
 - Constructed before their enclosing class objects (host objects)





Outline



1. Class definition

1.1 Member functions

1.2 Member variables

```
1 // Fig. 7.4: date1.h
2 // Declaration of the Date class.
3 // Member functions defined in date1.cpp
4 #ifndef DATE1_H
5 #define DATE1_H
6
7 class Date {
8 public:
9     Date( int = 1, int = 1, int = 1900 ); // default constructor
10    void print() const; // print date in month/day/year format
11
12 private:
13     int month; // 1-12
14     int day; // 1-31 based on month
15     int year; // any year
16
17    // utility function to test proper day for month and year
18    int checkDay( int );
19 };
20
21 #endif
```



Outline



1. Load header

1.1 Function definitions

1.2 Date constructor

```

22 // Fig. 7.4: date1.cpp
23 // Member function definitions for Date class.
24 #include <iostream>
25
26 using std::cout;
27 using std::endl;
28
29 #include "date1.h"
30
31 // Constructor: Confirm proper value for month;
32 // call utility function checkDay to confirm proper
33 // value for day.
34 Date::Date( int mn, int dy, int yr )
35 {
36     if ( mn > 0 && mn <= 12 )          // validate the month
37         month = mn;
38     else {
39         month = 1;
40         cout << "Month " << mn << " invalid. Set to month 1.\n";
41     }
42
43     year = yr;                          // should validate yr
44     day = checkDay( dy );               // validate the day
45
46     cout << "Date object constructor for date ";
47     print();                            // interesting: a print with no arguments
48     cout << endl;
49 }
50

```

Constructor will print
a line when called.



Outline



1.3 print function

1.4 Date destructor

1.5 checkDay function

```
51 // Print Date object in form month/day/year
52 void Date::print() const
53     { cout << month << '/' << day << '/' << year; }
54
55
56
57
58
59
60
61
62
63 // Utility function to confirm proper day value
64 // based on month and year.
65 // Is the year 2000 a leap year?
66 int Date::checkDay( int testDay )
67 {
68     static const int daysPerMonth[ 13 ] =
69         {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
70
71     if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
72         return testDay;
73
74     if ( month == 2 &&          // February: Check for leap year
75         testDay == 29 &&
76         ( year % 400 == 0 ||
77           ( year % 4 == 0 && year % 100 != 0 ) ) )
78         return testDay;
79
80     cout << "Day " << testDay << " invalid. Set to day 1.\n";
81
82     return 1; // leave object in consistent state if bad value
83 }
```



Outline



1. Load header

1.1 Class definition

1.2 Member functions

1.3 Member variables

1.3.1 Include const variables from Date class

```
84 // Fig. 7.4: empty1.h
85 // Declaration of the Employee class.
86 // Member functions defined in empty1.cpp
87 #ifndef EMPLOY1_H
88 #define EMPLOY1_H
89
90 #include "date1.h"
91
92 class Employee {
93 public:
94     Employee( char *, char *, int, int, int, int, int, int );
95     void print() const;
96
97 private:
98     char firstName[ 25 ];
99     char lastName[ 25 ];
100     const Date birthDate;
101     const Date hireDate;
102 };
103
104#endif
```

Composition - including
objects of other classes.



1. Load header files

1.1 Function definitions

1.2 Employee constructor

1.2.1 Use member-initializer syntax for const Date members

```

105// Fig. 7.4: emply1.cpp
106// Member function definitions for Employee class.
107#include <iostream>
108
109using std::cout;
110using std::endl;
111
112#include <cstring>
113#include "emply1.h"
114#include "date1.h"
115
116Employee::Employee( char *fname, char *lname,
117                    int bmonth, int bday, int byear,
118                    int hmonth, int hday, int hyear )
119    : birthDate( bmonth, bday, byear ),
120      hireDate( hmonth, hday, hyear )
121{
122    // copy fname into firstName and be sure that it fits
123    int length = strlen( fname );
124    length = ( length < 25 ? length : 24 );
125    strncpy( firstName, fname, length );
126    firstName[ length ] = '\0';
127
128    // copy lname into lastName and be sure that it fits
129    length = strlen( lname );
130    length = ( length < 25 ? length : 24 );
131    strncpy( lastName, lname, length );
132    lastName[ length ] = '\0';
133
134    cout << "Employee object constructor: "
135          << firstName << ' ' << lastName << endl;
136}

```

Constructor will print a line when called.



1.3 print definition

1.4 Employee destructor

137

138 `void Employee::print() const`

139 {

140 `cout << lastName << ", " << firstName << "\nHired: ";`141 `hireDate.print();`142 `cout << " Birth date: ";`143 `birthDate.print();`144 `cout << endl;`

145 }

146

147

148

149

150

151

152

The **print** function is **const** and will print whenever a **Date** object is created or destroyed. It can print **const** objects because it is a **const** function.

Print requires no arguments, it is linked implicitly to the object that calls it.



1. Load header files

2. Create Employee object

2.1 Attempt invalid Date setting

Only **employ.h** has to be loaded;
that file has the command to load
date.h.

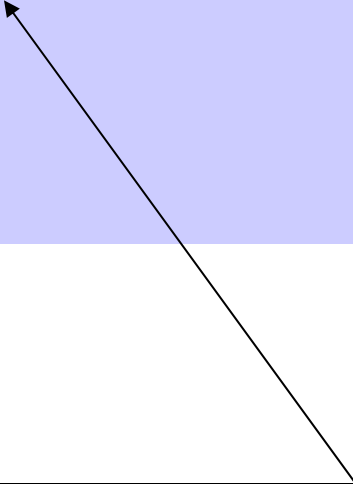
```
153// Fig. 7.4: fig07_04.cpp
154// Demonstrating composition: an object with member objects.
155#include <iostream>
156
157using std::cout;
158using std::endl;
159
160#include "employ1.h"
161
162int main()
163{
164    Employee e( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );
165
166    cout << '\n';
167    e.print();
168
169    cout << "\nTest Date constructor with invalid values:\n";
170    Date d( 14, 35, 1994 ); // invalid Date values
171    cout << endl;
172    return 0;
173}
```

**Program Output**

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

```
Jones, Bob  
Hired: 3/12/1988   Birth date: 7/24/1949
```

```
Test Date constructor with invalid values:  
Month 14 invalid. Set to month 1.  
Day 35 invalid. Set to day 1.  
Date object constructor for date 1/1/1994
```



Notice how inner objects are created