W6.2 Continuing Classes

- Classes with `const` qualifiers
- Class Composition or Aggregation

7.1 Introduction

- Chapters 6 through 8 discuss object-based programming (OBP)
- Chapters 9 and 10 discuss inheritance and polymorphism
7.2 const (Constant) Objects and const Member Functions

- 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 and initializes it to 12

- const objects require const functions
  - Member functions declared const cannot modify their object
  - const must be specified in function prototype and definition
  - Prototype:
    ReturnType FunctionName(param1,param2…) const;
  - Definition:
    ReturnType FunctionName(param1,param2…) const { …}
  - Example:
    int A::getValue() const { return privateDataMember };
    - Returns the value of a data member but doesn’t modify anything so is declared const

- Constructors / Destructors cannot be const
  - They need to initialize variables, therefore modifying them
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

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; }
1. Initialize variables

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

Compiler errors generated:

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.

Program Output

test.exe - 2 error(s), 0 warning(s)
7.2 const (Constant) Objects and const Member Functions

- Member initializer syntax
  - Data member increment in class Increment
  - constructor for Increment is modified as follows:
    
    ```cpp
    Increment::Increment( int c, int i )
    : increment( i )
    { count = c; }
    ```
  - `increment(i)` initializes increment to `i`
  - All data members can be initialized using member initializer syntax
  - consts and references must be initialized using member initializer syntax
  - Multiple member initializers
    - Use comma-separated list after the colon

If we try to initialize increment with an assignment statement (such as `increment = i`) instead of a member initializer we get an error.
34 Increment value(10, 5);
35 cout << "Before incrementing: ";
36 value.print();
37
38 for (int j = 0; j < 3; j++) {
39   value.addIncrement();
40   cout << "After increment " << j + 1 << ": ";
41   value.print();
42 }
43
44 return 0;
45 }
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    ~Date();  // provided to confirm destruction order
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

const char* public member function print();
1.4 Date destructor

1.5 checkDay function

Destructor will print a line when called.

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

Constructor will print a line when called.

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.

Destructor will print a line when called.
// Fig. 7.4: fig07_04.cpp
// Demonstrating composition: an object with member objects.
#include <iostream>

using std::cout;
using std::endl;

#include "emply1.h"

int main()
{
   Employee e( "Bob", "Jones", 7, 24, 1949, 3, 12, 1988 );

   cout << '
';
   e.print();

   cout << "Test Date constructor with invalid values:
";
   Date d( 14, 35, 1994 );  // invalid Date values
   cout << endl;
   return 0;
}