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
    
    ```cpp
    const Time noon(12, 0, 0);
    ```
    • Declares a const object noon of class Time and initializes it to 12
7.2 const (Constant) Objects and 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:
    
    \[ \text{ReturnType \ FunctionName(param1,param2... \ const;} \]
  
  - Definition:
    
    \[ \text{ReturnType \ FunctionName(param1,param2... \ const \{ \ ... \} \]
  
  - Example:
    
    \[ \text{int A::getValue() \ const \{ return \}
                          
                        \text{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
```cpp
// Fig. 7.1: time5.h
// Declaration of the class Time.
// Member functions defined in time5.cpp
#ifndef TIME5_H
#define TIME5_H

class Time {
public:
    Time( int = 0, int = 0, int = 0 ); // default constructor

    // set functions
    void setTime( int, int, int ); // set time
    void setHour( int ); // set hour
    void setMinute( int ); // set minute
    void setSecond( int ); // set second

    // get functions (normally declared const)
    int getHour() const; // return hour
    int getMinute() const; // return minute
    int getSecond() const; // return second

    // print functions (normally declared const)
    void printMilitary() const; // print military time
    void printStandard(); // print standard time
private:
    int hour; // 0 - 23
    int minute; // 0 - 59
    int second; // 0 - 59
};
#endif
```
// Fig. 7.1: time5.cpp
// Member function definitions for Time class.
#include <iostream>

using std::cout;

#include "time5.h"

// Constructor function to initialize private data.
// Default values are 0 (see class definition).
Time::Time( int hr, int min, int sec )
    { setTime( hr, min, sec ); }

// Set the values of hour, minute, and second.
void Time::setTime( int h, int m, int s )
    {
        setHour( h );
        setMinute( m );
        setSecond( s );
    }

// Set the hour value
void Time::setHour( int h )
    { hour = ( h >= 0 && h < 24 ) ? h : 0; }

// Set the minute value
void Time::setMinute( int m )
    { minute = ( m >= 0 && m < 60 ) ? m : 0; }

// Set the second value
void Time::setSecond( int s )
    { second = ( s >= 0 && s < 60 ) ? s : 0; }

The constructor is non-const but it can be called for const objects.
1.1 Function definitions

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

Keyword `const` in function definition and prototype.

Non-`const` functions cannot use `const` objects, even if they don’t modify them (such as `printStandard`).
// Fig. 7.1: fig07_01.cpp
// Attempting to access a const object with non-const member functions.
#include "time5.h"

int main()
{
    Time wakeUp( 6, 45, 0 );       // non-constant object
    const Time noon( 12, 0, 0 );   // constant object

    // MEMBER FUNCTION   OBJECT
    wakeUp.setHour( 18 );  // non-const         non-const
    noon.setHour( 12 );    // non-const         const

    wakeUp.getHour();      // const             non-const
    noon.getMinute();      // const             const
    noon.printMilitary();  // const             const
    noon.printStandard();  // non-const         const

    return 0;
}

Compiler errors generated.

Program Output

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)
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:
    ```
    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
  - `const`s and references must be initialized using member initializer syntax
  - Multiple member initializers
    - Use comma-separated list after the colon
1. Class definition

1.1 Function definitions

```cpp
// Fig. 7.2: fig07_02.cpp
// Using a member initializer to initialize a
// constant of a built-in data type.
#include <iostream>

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

class Increment {
public:
    Increment( int c = 0, int i = 1 );
    void addIncrement() { count += increment; }  
    void print() const;

private:
    int count;
    const int increment;   // const data member
};

// Constructor for class Increment
Increment::Increment( int c, int i )
    : increment( i ) // initializer for const member
    { count = c; } 

// Print the data
void Increment::print() const
    {
        cout << "count = " << count 
             << ", increment = " << increment << endl;
    }

int main()
    {
        // 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
36    cout << "Before incrementing: ";
37    value.print();
38
39    for ( int j = 0; j < 3; j++ ) {
40       value.addIncrement();
41       cout << "After increment " << j + 1 << ": ";
42       value.print();
43    }
44
45    return 0;
46 }
```

Before incrementing: count = 10, increment = 5
After increment 1: count = 15, increment = 5
After increment 2: count = 20, increment = 5
After increment 3: count = 25, increment = 5
7.3 Composition: Objects as Members of Classes

- **Composition**
  - 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)
// Fig. 7.4: date1.h
// Declaration of the Date class.
// Member functions defined in date1.cpp

#ifndef DATE1_H
#define DATE1_H

class Date {
public:
    Date( int = 1, int = 1, int = 1900 ); // default constructor
    void print() const;  // print date in month/day/year format
    ~Date();  // provided to confirm destruction order

private:
    int month;  // 1-12
    int day;    // 1-31 based on month
    int year;   // any year

    // utility function to test proper day for month and year
    int checkDay( int );
};
#endif
```cpp
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 }
```

Constructor will print a line when called.
51 // Print Date object in form month/day/year
52 void Date::print() const
53    { cout << month << '/' << day << '/' << year; }
54
55 // Destructor: provided to confirm destruction order
56 Date::~Date()
57 {
58    cout << "Date object destructor for date ";
59    print();
60    cout << endl;
61 }
62
63 // Utility function to confirm proper day value based on month and year.
64 // Is the year 2000 a leap year?
65 int Date::checkDay( int testDay )
66 {
67    static const int daysPerMonth[ 13 ] =
68       {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
69
70    if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
71       return testDay;
72
73    if ( month == 2 &&
74         testDay == 29 &&
75         ( year % 400 == 0 ||
76          ( year % 4 == 0 && year % 100 != 0 ) ) )
77       return testDay;
78
79    cout << "Day " << testDay << " invalid. Set to day 1.\n";
80
81    return 1; // leave object in consistent state if bad value
82}
// Fig. 7.4: emp1y1.h
// Declaration of the Employee class.
// Member functions defined in emp1y1.cpp
#ifndef EMPLY1_H
#define EMPLY1_H

#include "date1.h"

class Employee {
public:
    Employee( char *, char *, int, int, int, int, int, int );
    void print() const;
    ~Employee();  // provided to confirm destruction order

private:
    char firstName[ 25 ];
    char lastName[ 25 ];
    const Date birthDate;
    const Date hireDate;
};

#endif

Composition - including objects of other classes.
// Fig. 7.4: emply1.cpp
// Member function definitions for Employee class.
#include <iostream>

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

#include <cstring>
#include "emply1.h"
#include "date1.h"

Employee::Employee( char *fname, char *lname,
                    int bmonth, int bday, int byear,
                    int hmonth, int hday, int hyear )
   : birthDate( bmonth, bday, byear ),
     hireDate( hmonth, hday, hyear )
{
   // copy fname into firstName and be sure that it fits
   int length = strlen( fname );
   length = ( length < 25 ? length : 24 );
   strncpy( firstName, fname, length );
   firstName[ length ] = '\0';

   // copy lname into lastName and be sure that it fits
   length = strlen( lname );
   length = ( length < 25 ? length : 24 );
   strncpy( lastName, lname, length );
   lastName[ length ] = '\0';

   cout << "Employee object constructor: "
        << firstName << ' ' << lastName << endl;
}

Constructor will print a line when called.
void Employee::print() const
{
   cout << lastName << ", " << firstName << "\nHired: ";
   hireDate.print();
   cout << " Birth date: ";
   birthDate.print();
   cout << endl;
}

// Destructor: provided to confirm destruction order
Employee::~Employee()
{
   cout << "Employee object destructor: "
   << lastName << ", " << firstName << endl;
}

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;
}
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

Date object destructor for date 1/1/1994
Employee object destructor: Jones, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949

Notice how inner objects are created first and destroyed last.