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

- `const` objects require `const` functions
  - Member functions declared `const` cannot modify their object
  - `const` must be specified in function prototype and definition
    - Prototype:
      ```cpp
      ReturnType FunctionName(param1,param2...) const;
      ```
    - Definition:
      ```cpp
      ReturnType FunctionName(param1,param2...) const { ...}
      ```
  - Example:
    ```cpp
    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
### 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 )   // initializer for const member
      { count = c; }
      ```
    - `: increment( i )` initializes increment to `i`
    - All data members can be initialized using member initializer syntax
    - `const` and references must be initialized using member initializer syntax
    - Multiple member initializers
      - Use comma-separated list after the colon

### 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)
```cpp
135        << firstName << ' ' << lastName << endl;
130   length = ( length < 25 ? length : 24 );
128   // copy lname into lastName and be sure that it fits
127
125   strncpy( firstName, fname, length );
124   length = ( length < 25 ? length : 24 );
123   int length = strlen( fname );
122   // copy fname into firstName and be sure that it fits
121{
120     hireDate( hmonth, hday, hyear )
119   : birthDate( bmonth, bday, byear ),
118                    int hmonth, int hday, int hyear )
117                    int bmonth, int bday, int byear,
112#include <cstring>
111
110using std::endl;
109using std::cout;
108
106// Member function definitions for Employee class.
105// Fig. 7.4: emply1.cpp
104#endif
103
102};
101   const Date hireDate;
100   const Date birthDate;
99    char lastName[ 25 ];
98    char firstName[ 25 ];
97 private:
96    ~Employee();  // provided to confirm destruction order
94    Employee( char *, char *, int, int, int, int, int, int ); // default constructor
93 public:
92 class Employee {
90 #include "date1.h"
88 #define EMPLY1_H
87 #ifndef EMPLY1_H
86 // Member functions defined in emply1.cpp
85 // Declaration of the Employee class.
84 // Fig. 7.4: emply1.h
83
82    return 1;  // leave object in consistent state if bad value
80    cout << "Day " << testDay << " invalid. Set to day 1."
79
78       return testDay;
77          ( year % 4 == 0 && year % 100 != 0 ) )
76         ( year % 400 == 0 ||
75
70    static const int daysPerMonth[ 13 ] =
68    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
66 int Date::checkDay( int testDay )
65 // Is the year 2000 a leap year?
64 // based on month and year.
63 // Utility function to confirm proper day value
62
61   return testDay;
60    print();
59    int checkDay( int );
58    // utility function to test proper day for month and year
57
56 Date::~Date()
55 // Destructor: provided to confirm destruction order
54
53    { cout << month << '/' << day << '/' << year; }
52 void Date::print() const
51 // Print Date object in form  month/day/year
50    day = checkDay( dy );           // validate the day
49    year = yr;                      // should validate yr
48    month = 1;                      // set default month
47    cout << "Month " << mn << " invalid. Set to month 1."
46    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
44    if ( mn > 0 && mn <= 12 )       // validate the month
43    { Date::Date( int mn, int dy, int yr )
42 // value for day.
41    } else {
40       month = 1;
39       month = 1;
38    else {
37    } else {
36    if ( mn > 0 && mn <= 12 )       // validate the month
35    { Date::Date( int mn, int dy, int yr )
34 // value for day.
33 // Constructor: Confirm proper value for month;
32
31 // Member functions defined in date1.cpp
30
29 #include "date1.h"
28
26 using std::cout;
25 // Member function definitions for Date class.
24 // Fig. 7.4: date1.cpp
23
22 int Date::checkDay( int testDay )
21 // Is the year 2000 a leap year?
20 // based on month and year.
21 // Utility function to confirm proper day value
20
19 };
```cpp
#include <iostream>

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

#include "employee.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;
}
```

1. Load header files
2. Create Employee object
2.1 Attempt invalid date setting

Only `employee.h` has to be loaded. The file has the command to load `date.h`.

Program Output

- Date object constructor for date 7/24/1949
- Employee object constructor: Bob Jones
- Hired: 3/12/1988  Birth date: 7/24/1949
- Date object constructor for date 3/12/1988
- 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.