Performance Tip 10.1

The extra object that's created by the postfix increment (or decrement) operator can result in a performance problem—especially when the operator is used in a loop. For this reason, you should prefer the overloaded prefix increment and decrement operators.

10.8 Case Study: A `Date` Class

- The program of Figs. 10.6–10.8 demonstrates a `Date` class, which uses overloaded prefix and postfix increment operators to add 1 to the day in a `Date` object, while causing appropriate increments to the month and year if necessary.

```
1  // Fig. 10.6: Date.h
2  // Date class definition with overloaded increment operators.
3  #ifndef DATE_H
4  #define DATE_H
5
6  #include <array>
7  #include <iostream>
8
9  class Date
10 {
11    friend std::ostream &operator<<( std::ostream &, const Date & );
12 public:
13    Date( int m = 1, int d = 1, int y = 1900 ); // default constructor
14    void setDate( int, int, int ); // set month, day, year
15    Date &operator++(); // prefix increment operator
16    Date operator++( int ); // postfix increment operator
17    Date &operator+=( unsigned int ); // add days, modify object
18    static bool leapYear( int ); // is date in a leap year?
19    bool endOfMonth( int ) const; // is date at the end of month?
20 private:
21    unsigned int month;
22    unsigned int day;
23    unsigned int year;
```

Fig. 10.6 | `Date` class definition with overloaded increment operators. (Part 1 of 2.)

```
24
25      static const std::array< unsigned int, 13 > days; // days per month
26      void helpIncrement(); // utility function for incrementing date
27   }; // end class Date
28
29   #endif
```

Fig. 10.6 | `Date` class definition with overloaded increment operators. (Part 2 of 2.)

```
1  // Fig. 10.7: Date.cpp
2  // Date class member- and friend-function definitions.
3  #include <iostream>
4  #include <string>
5  #include "Date.h"
6  using namespace std;
7
8  // initialize static member; one classwide copy
9  const array< unsigned int, 13 > Date::days =
10    { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
11
12 // Date constructor
13 Date::Date( int month, int day, int year )
14 {
15    setDate( month, day, year );
16 } // end Date constructor
17
```

Fig. 10.7 | Date class member- and friend-function definitions. (Part 1 of 6.)

```
18  // set month, day and year
19  void Date::setDate( int mm, int dd, int yy )
20  {
21     if ( mm >= 1 && mm <= 12 )
22        month = mm;
23     else
24        throw invalid_argument( "Month must be 1-12" );
25
26     if ( yy >= 1900 && yy <= 2100 )
27        year = yy;
28     else
29        throw invalid_argument( "Year must be >= 1900 and <= 2100" );
30
31     // test for a leap year
32     if ( ( month == 2 && leapYear( year ) && dd >= 1 && dd <= 29 ) ||
33          ( dd >= 1 && dd <= days[ month ] ) )
34        day = dd;
35     else
36        throw invalid_argument(
37           "Day is out of range for current month and year" );
38  } // end function setDate
39
```

Fig. 10.7 | `Date` class member- and `friend`-function definitions. (Part 2 of 6.)

```
40 // overloaded prefix increment operator
41 Date &Date::operator++()
42 {
43    helpIncrement(); // increment date
44    return *this; // reference return to create an lvalue
45 } // end function operator++
46
47 // overloaded postfix increment operator; note that the
48 // dummy integer parameter does not have a parameter name
49 Date Date::operator++( int )
50 {
51    Date temp = *this; // hold current state of object
52    helpIncrement();
53
54    // return unincremented, saved, temporary object
55    return temp; // value return; not a reference return
56 } // end function operator++
57
```

Fig. 10.7 | `Date` class member- and `friend`-function definitions. (Part 3 of 6.)

```
58 // add specified number of days to date
59 Date &Date::operator+=( unsigned int additionalDays )
60 {
61    for ( int i = 0; i < additionalDays; ++i )
62       helpIncrement();
63
64    return *this; // enables cascading
65 } // end function operator+=
66
67 // if the year is a leap year, return true; otherwise, return false
68 bool Date::leapYear( int testYear )
69 {
70    if ( testYear % 400 == 0 ||
71       ( testYear % 100 != 0 && testYear % 4 == 0 ) )
72       return true; // a leap year
73    else
74       return false; // not a leap year
75 } // end function leapYear
76
```

Fig. 10.7 | `Date` class member- and `friend`-function definitions. (Part 4 of 6.)

```
77  // determine whether the day is the last day of the month
78  bool Date::endOfMonth( int testDay ) const
79  {
80     if ( month == 2 && leapYear( year ) )
81        return testDay == 29; // last day of Feb. in leap year
82     else
83        return testDay == days[ month ];
84  } // end function endOfMonth
85
86  // function to help increment the date
87  void Date::helpIncrement()
88  {
89     // day is not end of month
90     if ( !endOfMonth( day ) )
91        ++day; // increment day
92     else
93        if ( month < 12 ) // day is end of month and month < 12
94        {
95           ++month; // increment month
96           day = 1; // first day of new month
97        } // end if
```

Fig. 10.7 | `Date` class member- and `friend`-function definitions. (Part 5 of 6.)

```
98         else // last day of year
99         {
100           ++year; // increment year
101           month = 1; // first month of new year
102           day = 1; // first day of new month
103        } // end else
104  } // end function helpIncrement
105
106  // overloaded output operator
107  ostream &operator<<( ostream &output, const Date &d )
108  {
109     static string monthName[ 13 ] = { "", "January", "February",
110        "March", "April", "May", "June", "July", "August",
111        "September", "October", "November", "December" };
112     output << monthName[ d.month ] << ' ' << d.day << ", " << d.year;
113     return output; // enables cascading
114  } // end function operator<<
```

Fig. 10.7 | `Date` class member- and `friend`-function definitions. (Part 6 of 6.)

```
1  // Fig. 10.8: fig10_08.cpp
2  // Date class test program.
3  #include <iostream>
4  #include "Date.h" // Date class definition
5  using namespace std;
6
7  int main()
8  {
9     Date d1( 12, 27, 2010 ); // December 27, 2010
10    Date d2; // defaults to January 1, 1900
11
12    cout << "d1 is " << d1 << "\nd2 is " << d2;
13    cout << "\n\nd1 += 7 is " << ( d1 += 7 );
14
15    d2.setDate( 2, 28, 2008 );
16    cout << "\n\n  d2 is " << d2;
17    cout << "\n++d2 is " << ++d2 << " (leap year allows 29th)";
18
19    Date d3( 7, 13, 2010 );
20
21    cout << "\n\nTesting the prefix increment operator:\n"
22       << "  d3 is " << d3 << endl;
23    cout << "++d3 is " << ++d3 << endl;
24    cout << "  d3 is " << d3;
```

Fig. 10.8 | Date class test program. (Part 1 of 2.)

```
25
26      cout << "\n\nTesting the postfix increment operator:\n"
27         << "  d3 is " << d3 << endl;
28      cout << "d3++ is " << d3++ << endl;
29      cout << "  d3 is " << d3 << endl;
30   } // end main
```

```
d1 is December 27, 2010
d2 is January 1, 1900

d1 += 7 is January 3, 2011

  d2 is February 28, 2008
++d2 is February 29, 2008 (leap year allows 29th)

Testing the prefix increment operator:
  d3 is July 13, 2010
++d3 is July 14, 2010
  d3 is July 14, 2010

Testing the postfix increment operator:
  d3 is July 14, 2010
d3++ is July 14, 2010
  d3 is July 15, 2010
```

Fig. 10.8 | Date class test program. (Part 2 of 2.)

10.8 Case Study: A `Date` Class (cont.)

- The `Date` constructor (defined in Fig. 10.7, lines 13–16) calls `setDate` to validate the month, day and year specified.
  - Invalid values for the month, day or year result in `invalid_argument` exceptions.

10.8 Case Study: A `Date` Class (cont.)

Date Class Prefix Increment Operator

- Overloading the prefix increment operator is straightforward.
  - The prefix increment operator (defined in Fig. 10.7, lines 41–45) calls utility function `helpIncrement` (defined in Fig. 10.7, lines 87–104) to increment the date.
  - This function deals with "wraparounds" or "carries" that occur when we increment the last day of the month.
  - These carries require incrementing the month.
  - If the month is already 12, then the year must also be incremented and the month must be set to 1.
  - Function `helpIncrement` uses function `endOfMonth` to determine whether the end of a month has been reached and increment the day correctly.

10.8 Case Study: A `Date` Class (cont.)

- The overloaded prefix increment operator returns a reference to the current `Date` object (i.e., the one that was just incremented).
- This occurs because the current object, `*this`, is returned as a `Date &`.
  - Enables a preincremented `Date` object to be used as an *lvalue,* which is how the built-in prefix increment operator works for fundamental types.