UNBC Laboratory Assignment
Computer Science 101—Winter 2007

Overloading Operators for the `Date` Class

Due Date:

This assignment is due 12 March 2007 *at the beginning of lecture.*

Estimated Time:

three hours.

Overloaded Operators

Programming exercises:

Using your previously coded `Date` class,

- § add overloaded operator member functions for • `+=` (with an `int` right-hand-side argument) • `-=` (as for `+=`) • `++` (prefix) • `++` (postfix) • `--` (prefix) • `--` (postfix) • `-` (with a `Date` right-hand-side argument returning the number of days between left- and right-hand-side arguments) and • `-` (with a `in` right-hand-side argument returning a `Date`);
- § add overloaded operator non-member functions for • `+` (both `int` plus `Date`, and `Date` plus `int`); • `<` • `>` • `<=` • `>=` • `!=` • `==`
- § add overloaded operator non-member functions for • `<<` and • `>>`.

Hints and Commandments

Some of the overloaded operator member functions can be coded in terms of member functions that you have previously written. Use these previously written member functions if you can. For instance, it is likely that you can write the `Date::operator+=` function as[1]

```
Date& Date::operator+=(int days) {   addDays(days) ; return *this ; }
```

The += and −= functions should return a `Date&`. The + and − operators should return `Date` objects, that is, not references (excepting the `Date - Date` operator which should return `int`).

Carefully read Chapter 8 of *Deitel and Deitel* before attempting to code the ++ and −− operators. Write the postfix versions of these operators to call the prefix versions.

[+] This **.pdf** created February 25, 2007.

[1] the code here is condensed for space reason; not as a recommendation for good coding style.

More generally, it is good practice to code overloaded operators in terms of other overloaded operators. For instance, one can code + using +=, and this is good programming practice because it guarantees that + using += remain logically consistent.

(The more general programming precept here is *never repeat code*, as repeated code is an invitation to maintenance mistakes, when one version of the repeated code is changed, and the other is not.)

Continuing in this vein, it is a good idea to exploit the mathematical tautologies $a > b \Leftrightarrow b < a$, $a \geq b \Leftrightarrow \neg(a < b)$, $a \neq b \Leftrightarrow \neg(a = b)$, and so on. For instance, one can write

```
bool operator!=(Date const& lhs, Date const& rhs) {return !(lhs==rhs);}
```

provided that one has already overloaded the == operator.

The << operator must be coded as a non-member function for reasons discussed in *Deitel and Deitel*.

The >> operator must be coded as a non-member function for reasons discussed in *Deitel and Deitel*. This is probably the hardest operator to code well.

Further Programming exercises:

Use your previous test/driver programs as a starting point to create a longer test package for your class.

Add one more member function to your class, `Date::dayOfTheWeek`, which returns 0 for Sunday, 1 for Monday, and so on. *Think hard* before doing this. You should be able to code this function as a one-liner!