COMSC-200 Lecture Topic 8
Operator Overloading

Reference
Deitel Ch.11.1-11.8,11.10

Operator Overloading
define +, ==, and other operators
as they apply to objects of *your* class
example:
`Time t(9, 34, 56);`
what does `t += 10;` mean?

Why Overload Operators?
why not just use functions?
example: `t.addSecs(10);` *or* `addSecs(t, 10);`
because:
using classes with templates:
template code uses operators
makes programs more readable
it's the C++ way

Considerations
cannot define new operators
must match unary-ness, binary-ness
should match logically
not `:: ?: .` or `sizeof`

operator==
return `bool` value
example: *(const data members okay)*

```
bool operator==(const Time&) const; // prototype
bool Time::operator==(const Time& t) const
{
  return (hour == t.hour
    && minute == t.minute
    && second == t.second);
}
```

or it can be a friend function

```
bool operator==(const Time& a, const Time& b)
{
  return (a.hour == b.hour
    && a.minute == b.minute
    && a.second == b.second);
}
```

operator+
return `Time` object (new copy)
can have any parameter data type
overloading possible
example:

```
Time operator+(int) const; // prototype
Time Time::operator+(int n) const
{
  return Time(hour, minute, second + n);
}
```

or it can be a friend function

operator+=
return `*this` (mutable self-reference)
example:

```
Time& operator+=(int); // prototype
Time& Time::operator+=(int n)
{
  second += n; // not const
  return *this;
}
```

operator<<
cannot be a member function
usually a `friend` for better performance
example:

```
#include<iostream>
using std::ostream;
friend ostream& operator<<(ostream&, const Time&);
ostream& operator<<(ostream& out, const Time& t)
{
  out << t.hour << ':'
    << t.minute << ':' << t.second;
  return out;
}
```

type conversion
use for casting (implicit and explicit)
example:

```
operator int() const; // prototype
Time::operator int() const
{
  return 3600 * hour + 60 * minute + second;
}
```

operator=
the "assignment" operator
req'd when dynamic memory used in class
or const members and assignment used
return `*this` (mutable self-reference)
option: returning `const` prevents `(a = b) = c`
example: *(no const data members)*

```
Time& operator=(const Time&); // prototype
Time& Time::operator=(const Time& t)
{
  if (this == &t) return *this;

  hour = t.hour;
  minute = t.minute;
  second = t.second;

  return *this;
}
```

used in `a = b;`
actually not used in `Time a = b;`
uses copy contructor, if available
default assignment does field-by-field copy
but still uses operator=
of member objects that have one

Operator Overloading Key

Operator Type	Function Type	Return Type
comparison (`operator==`, etc)	getter member, friend, or stand-alone	`bool`

arithmetic (`operator+`, etc)	getter member, friend, or stand-alone	new object copy
assignment (`operator=`)	setter member	mutable self reference
compound assignment (`operator+=`, etc)	setter member	mutable self reference
stream insertion (`operator<<`)	friend or stand-alone	mutable `ostream` reference
type conversion (`operator int()`, etc)	getter member	*none*, but do return matching value

---

Example: Static Object Arrays As Class Databases

```
class Year
{
  static const int NHOLIDAYS; // size of HOLIDAY array
  static const Date HOLIDAY[]; // the holidays of a year
  ...
  static void listHolidays() const
  {
    for (int i = 0; i < NHOLIDAYS; i++)
      HOLIDAY[i].list();
  }
};

const int Year::NHOLIDAYS = 7; // or assign upon declaration inside class
const Date Year::HOLIDAY[] = // database
{
  Date(1, 1, "New Year's Day"),
  Date(1, 25, "Robert Burns Day"),
  Date(2, 17, "Saint Valentine's Day"),
  Date(6, 14, "Flag Day"),
  Date(7, 4, "Independence Day"),
  Date(11, 11, "Veterans Day"),
  Date(12, 25, "Christmas Day")
};
```

Elevator Simulation Index: where to find stuff...

class Rider, lab 8	class Elevator, lab 9	class Floor, lab 14
Rider::operator=, lecture 9	Elevator EXEs, lecture 9	Floor::addNewRider, lecture 14
Elevator::isNearDestination, lab 9	Elevator::moveToDestinationFloor, lab 9	Floor::isPreferredDirectionUp, lecture 14
class Building, lab 15	Elevator::addRiders, lecture 10	Floor::removeDownRiders, lecture 14
getDifferentInts(int, int&, int&), lab 14	Elevator::removeRidersForDestinationFloor, lecture 10	Floor::removeUpRiders, lecture 14
poissonHits(double), lab 15	Elevator::setDestinationBasedOnRiders, lecture 10	GUI simulation option, lab 15