Operator overloading - how?

- Operators which can be overloaded - see list in text
  - Unary ops - one operand
  - Binary ops - two operands

Syntax:

```cpp
return-value operator (parameter list) {
  // source code for operator actions
} // end of function
```

Example:

```cpp
struct complex {
  float r, i;
};
complex operator + (complex a, complex b) {
  complex temp;
  temp.r = a.r + b.r;
  temp.i = a.i + b.i;
  return temp;
}
```

Note: the addition operator, +, is a binary operator.

Example

excerpted from book example - Time

```cpp
class Time {
public:
  Time (unsigned c = 0); // constructors
  // operator methods to set hours & minutes
  const Time& operator += ( ); // pre-increment
  Time operator ++( ); // post-increment (op. defined)
  // rest of class members |
}; // end of class Time
```
// Copy constructor: no dynamic memory - no delete
Time::Time(const Time& aTime) {
    minutes = aTime.minutes;
    hours = aTime.hours;
}

// Unary operator +=
const Time& Time::operator += (unsigned n) {
    unsigned t = minutes + n;
    minutes = t % (MAXMIN + 1);
    hours = t / (MAXMIN + 1);
    return *this;
}

// Unary operator - pre-increment
const Time& Time::operator ++() {
    if (++minutes > MAXMIN) {
        minutes = 0;
        ++hours; // call post-increment
    }
    return *this;
}

// Unary operator - post-increment
Time Time::operator ++(int)
{
    Time save = *this; // copy time to temp
    operator ++(); // call pre-increment op
    return save;
}

Some rules and factors

- Many ways to implement - some better than others
- Examine lots of code to see ways and why
- No default arguments
- Can not revise intrinsic variable operators
- May be nonstatic member(implicit) or friend function(explicit)
- There is usually a good reason for selection
- Operators +, *, [ ], -> must be member functions(implicit)
- One "argument" must be instance of class, reference to class, enumeration or reference to enumeration
Some rules and factors (cont’d)

- Operators new and delete can be overloaded
  check for the parameter types
- Operators not overloaded
  member access, '.'; access-deref., '.*'; scope, '::';
arithmetic if, '? :'
- Operators - only ones from the C++ precedence chart
- Precedence, grouping and number of operands can not be changed

Operator ideas for implicit or explicit

- No exact rule for selecting implicit vs explicit
- Operators +, ( ), [ ], -> must be member functions(implicit)
- Implicit preferred because member function vs friend
- Explicit when member function results in awkward expression

Example: output stream operator <<
  implicit - ostream& operator << (ostream& out);
  explicit - friend ostream& operator << (ostream& out, Point& p);
Point p; p.operator <<(cout); // implicit call
    cout << p; // explicit call

Case study 4 - VString

Implementation by operators as opposed to methods

You can add others - some occur by implicit conversion

String class operators we will examine
1. Assignment: =  e.g. stg1 = "Name"; stg1 = stg2;
   stg1 = 'c';
2. Concatenate +  e.g. stg1+stg2: stg1="Name";
   "Name"+ stg1;
3. Logical   => e.g. stg1 => stg2
   >=; <=; >::; ::; |(|IsEmpty)
4. Subscript [] e.g. ch = stg1[6];

Note: For a particular class, many operators do not make sense