```
//===================================
// Section 14 Operator overloading
//===================================

// What is operator overloading
- using traditional operators such as +,=,* with user defined types
- Allows our types to behave like built in ones
- Can make code more readable\writable
- Not done automatically except for the = sign, we must do the others ourselves

// Example
Without operator overloading, we might have to do something like this:
Number result = multiply(add(a,b), divide(c,d)); // Using member methods to do these calculations is cumbersome

With operator overloading, we can instead do this:
Number result = (a+b)*(c/d);

// What operators can be overloaded?
Most of them, except:
::  :?  .*  .   sizeof

// Basic rules
Precedence and associativity can't be changed
'arity' can't be changed - unary will be unary no matter what
Can't overload the operators for the primitive types
Can't create new operators
[], (), -> and the assignment operator (=) MUST be declared as member methods
Other operators can be member methods or global functions

// Examples
We can use the + operator on strings and on ints

// Overloading the assignment operator
Copy assignment (=)
The default operator used for assigning one object to another
Default is memberwise assignment (shallow copy)
- If we have a raw pointer we must deep copy

Assignment occurs when an object has already been initialized
Mystring s1{"Frank"};   // initialize an object
Mystring s2 = s1;       // We don't actually have s2 created yet, so this is an initialization using the copy constructor
s1 = s2;                // We already have an s1 so now we have assigned s2 to it

// How to overload the copy assignment operator
Type &Type::operator=(const Type &rhs); // Use the keyword operator followed by the operator we wish to overload

Mystring &Mystring::operator=(const Mystring &rhs);

// Behind the scenes the compiler converts this statement:
s2 = s1;

// to
s2.operator=(s1);

// Overloading the copy assignment operator with a deep copy
Mystring &Mystring::operator=(const Mystring &rhs){
    if (this == &rhs)
        return this;
    delete [] str;
    str = new char[std::strlen(thd.str) + 1];
    std::strcpy(str, rhs.str);

    return *this;
}
```

```
// Inheritance
What is it and why?
- Create new classes from existing classes
- New class contains all data and behavior of existing class
- Reuse of existing classes
- Focus on the common attributes among a set of classes
- New classes can modify the behavior of existing classes to make something unique without modifying the base class

// Related classes
Player, Enemy, Level Boss, Hero, Super Player, etc
- Identify things different objects might have in common

Account
- balance, deposit, withdraw

Savings Account
- balange, deposit, withdraw, interest rate

Checking Account
- balance, deposit, withdraw, minimum balance, per check fee

We can see there are lots of duplicated items. Compress the common ones into the base class

// Terminology and notation
Inheritance
- Process of creating new classes fro existing classes
- Reuse mechanism

Single Inheritance
- A new class created from another single class

Multiple Inheritance
- A new class is created from two or more other classes

Base Class (aka Parent class, superclass)
- The class being extended or inherited from

Derived class(child class, subclass)
- The class being created from the base class
- Will inherit attributes and operations from the base class

// Relationships
Is-a relationship
- Public inheritance
- Derived classes are sub-types of their base classes
- Can use a derived class object wherever we use a base class object

Generalization
- Combining similar classes into a single, more general class based on common attributes

Specialization
- Creating new classes from existing classes providing more specialized attributes or operations

Inheritance or class hierarchies
- Organization of the inheritance relationships
```