14.14 Assume there is a class named Pet. Write the prototype for a member function of Pet that overloads the = operator.

```
class Pet {
public:
  bool operator=(const Pet&);
};
```

14.15 Assume that dog and cat are instances of the Pet class, which has overloaded the = operator. Rewrite the following statement so it appears in function call notation instead of operator notation:

```
dog = cat;
dog.operator=(cat);
```

14.16 What is the disadvantage of an overloaded = operator returning void?

It doesn't compile. Don't argue, I checked. It doesn't even make any sense, since if that operator is ever used in an expression the compiler will try and coalesce that into a bool, and it makes absolutely not logical sense to try and cast void to bool.

14.17 Describe the purpose of the this pointer.

The this pointer allows you to reference the current object, used in resolving scope collisions and so forth. Modern JavaScript developers find this inferior to the _this convention, although their non-JS colleagues note this is a side-effect of their desecration of the sanctity of bracketed scope religion set forth in the Pentateuch.

14.18 The this pointer is automatically passed to what type of functions?

Member functions.

14.19 Assume there is a class named Animal that overloads the = and + operators. In the following statement, assume cat, tiger, and wildcat are all instances of the Animal class:

```
wildcat = cat + tiger;
```

Of the three objects, wildcat, cat, or tiger, which is calling the operator+ function? Which object is passed as an argument into the function?

cat is the object on which operator+ is being called with tiger as the argument.

14.20 What does the use of a dummy parameter in a unary operator function indicate to the compiler?

It indicates that the method is to be used in a postfix fashion.

14.21 Describe the values that should be returned from functions that overload relational operators.

It should always be either `true` or `false` (a `bool`).

14.22 What is the advantage of overloading the << and >> operators?

They let you play with streams, a language syntax that was included presumably because Bjarne Stroustrup felt that bit shift operators hadn't been abused enough by modern languages.

14.23 What type of object should an overloaded << operator function return?

The left-hand operand.

14.24 What type of object should an overloaded >> operator function return?

The left-hand operand.

14.25 If an overloaded << or >> operator accesses a private member of a class, what must be done in that class's declaration?

Declare them as `friend` functions.

14.26 Assume the class `NumList` has overloaded the `[]` operator. In the expression below, `list1` is an instance of the `NumList` class:

```
list1[25]
```

Rewrite the expression above to explicitly call the function that overloads the `[]` operator.

```
list1.operator[](25);
```

14.27 When overloading a binary operator such as + or –, what object is passed into the operator function's parameter?

The right-hand value of the expression.

14.28 Explain why overloaded prefix and postfix ++ and –– operator functions should return a value.

If they don't return a value, it will break consistency with the C-style of returning values from pre- and post-increment/decrement operators. Most of C++'s ugly baggage is from C.

14.29 How does C++ tell the difference between an overloaded prefix and postfix ++ or –– operator function?

The presence of a dummy argument.

14.30 Write member functions of the `FeetInches` class that overload the prefix and postfix –– operators. Demonstrate the functions in a simple program similar to Program 14-14.

```
class FeetInches {
public:
  FeetInches operator--() { // prefix
    inches--;
    simplify();
    return this;
  }
  FeetInches operator--(int) { // postfix
    FeetInches t(feet,inches);
    inches--;
    simplify();
    return t;
  }
};

int main() {
  double d;
  int i;

  FeetInches distance;

  cout << “Enter a distance in feet and inches:\n”;
  cin >> distance;

  d = distance;

  i = distance;

  cout << “The value “ << distance;
  cout << “ is equivalent to “ << d << “ feet\n”;
  cout << “ or “ << i << “ feet, rounded down. \m”;

  cout << “The distance “ << distance-- << “ less one inch is “ <<
distance << endl;
  cout << “The distance “ << distance << “ less yet another inch is “
<< --distance << endl;

  return 0;
}
```