In a clear departure from C and C++, Java and C# allow mixed-mode assignment only if the required coercion is widening. So, an int value can be assigned to a float variable, but not vice versa. Disallowing half of the possible mixed-mode assignments is a simple but effective way to increase the reliability of Java and C#, relative to C and C++.

In all languages that allow mixed-mode assignment, the coercion takes place only after the right side expression has been evaluated. One alternative would be to coerce all operands in the right side to the type of the target before evaluation. For example, consider the following code:

```java
int a, b;
float c;
...
c = a / b;
```

Because c is float, the values of a and b could be coerced to float before the division, which could produce a different value for c than if the coercion were delayed (for example, if a were 2 and b were 3).

**SUMMARY**

Expressions consist of constants, variables, parentheses, function calls, and operators. Assignment statements include target variables, assignment operators, and expressions.

The semantics of an expression is determined in large part by the order of evaluation of operators. The associativity and precedence rules for operators in the expressions of a language determine the order of operator evaluation in those expressions. Operand evaluation order is important if functional side effects are possible. Type conversions can be widening or narrowing. Some narrowing conversions produce erroneous values. Implicit type conversions, or coercions, in expressions are common, although they eliminate the error-detection benefit of type checking, thus lowering reliability.

Assignment statements have appeared in a wide variety of forms, including conditional targets and assigning operators.

**REVIEW QUESTIONS**

1. Define operator precedence and operator associativity.
2. Define functional side effect.
3. What is a coercion?
4. What is a conditional expression?
5. What is an overloaded operator?
6. Define narrowing and widening conversions.
7. What is a mixed-mode expression?
8. How does operand evaluation order interact with functional side effects?
9. What is short-circuit evaluation?
10. Name a language that always does short-circuit evaluation of Boolean expressions. Name one that never does it. Name one in which the programmer is allowed to choose.
11. How does C support relational and Boolean expressions?
12. What is the purpose of a compound assignment operator?
13. What is the associativity of C's unary arithmetic operators?
14. What is one possible disadvantage of treating the assignment operator as if it were an arithmetic operator?
15. What mixed-mode assignments are allowed in Ada?
16. What mixed-mode assignments are allowed in Java?

**PROBLEM SET**

1. When might you want the compiler to ignore type differences in an expression?
2. State your own arguments for and against allowing mixed-mode arithmetic expressions.
3. Do you think the elimination of overloaded operators in your favorite language would be beneficial? Why or why not?
4. Would it be a good idea to eliminate all operator precedence rules and require parentheses to show the desired precedence in expressions? Why or why not?
5. Should C's assigning operations (for example, +=) be included in other languages? Why or why not?
6. Should C's single-operand assignment forms (for example, ++count) be included in other languages? Why or why not?
7. Describe a situation in which the add operator in a programming language would not be commutative.
8. Describe a situation in which the add operator in a programming language would not be associative.
9. Assume the following rules of associativity and precedence for expressions:
**Programming Exercises**

1. Run the code given in Problem 13 (in the Problem Set) on some system that supports C to determine the values of `sum1` and `sum2`. Explain the results.
2. Rewrite the program of Exercise 1 in C++. Java, and C#. Run them, and compare the results.
3. Write a test program in your favorite language that determines and outputs the precedence and associativity of its arithmetic and Boolean operators.
4. Write an Ada program that illustrates the difference between `mod` and `rem` using a variety of positive and negative integer operands.
5. Write a Java program that exposes Java’s rule for operand evaluation order when one of the operands is a method call.
6. Repeat Exercise 5 with C++.
7. Repeat Exercise 6 with C#.