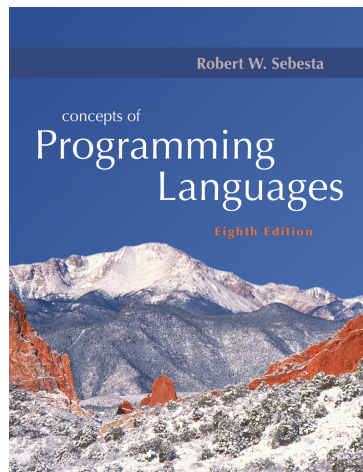


Chapter 7

Expressions and Assignment Statements



ISBN 0-321-49362-1

Chapter 7 Topics

- Introduction
- Arithmetic Expressions
- Overloaded Operators
- Type Conversions
- Relational and Boolean Expressions
- Short-Circuit Evaluation
- Assignment Statements
- Mixed-Mode Assignment

2

Introduction

- Expressions are the fundamental means of specifying computations in a programming language
- To understand expression evaluation, need to be familiar with the orders of operator and operand evaluation
- Essence of imperative languages is dominant role of assignment statements

3

Arithmetic Expressions

- Arithmetic evaluation was one of the motivations for the development of the first programming languages
- Arithmetic expressions consist of operators, operands, parentheses, and function calls

4

Arithmetic Expressions: Design Issues

- Design issues for arithmetic expressions
 - Operator precedence rules?
 - Operator associativity rules?
 - Order of operand evaluation?
 - Operand evaluation side effects?
 - Operator overloading?
 - Type mixing in expressions?

5

Arithmetic Expressions: Operators

- A unary operator has one operand
- A binary operator has two operands
- A ternary operator has three operands

6

Arithmetic Expressions: Operator Precedence Rules

- The operator precedence rules for expression evaluation define the order in which “adjacent” operators of different precedence levels are evaluated
- Typical precedence levels
 - parentheses
 - unary operators
 - ** (if the language supports it)
 - *, /
 - +, -

7

Arithmetic Expressions: Operator Associativity Rule

- The operator associativity rules for expression evaluation define the order in which adjacent operators with the same precedence level are evaluated
- Typical associativity rules
 - Left to right, except **, which is right to left
 - Sometimes unary operators associate right to left (e.g., in FORTRAN)
- APL is different; all operators have equal precedence and all operators associate right to left
- Precedence and associativity rules can be overridden with parentheses

8

Ruby Expressions

- All arithmetic, relational, and assignment operators, as well as array indexing, shifts, and bit-wise logic operators, are implemented as methods
 - One result of this is that these operators can all be overridden by application programs

9

Arithmetic Expressions: Conditional Expressions

- Conditional Expressions
 - C-based languages (e.g., C, C++)
 - An example:

```
average = (count == 0) ? 0 : sum / count
```
 - Evaluates as if written like

```
if (count == 0)
    average = 0
else
    average = sum / count
```

10

Arithmetic Expressions: Operand Evaluation Order

- Operand evaluation order
 1. Variables: fetch the value from memory
 2. Constants: sometimes a fetch from memory; sometimes the constant is in the machine language instruction
 3. Parenthesized expressions: evaluate all operands and operators first
 4. The most interesting case is when an operand is a function call

11

Arithmetic Expressions: Potentials for Side Effects

- Functional side effects: when a function changes a two-way parameter or a non-local variable
- Problem with functional side effects:
 - When a function referenced in an expression alters another operand of the expression; e.g., for a parameter change:

```
a = 10;
/* assume that fun changes its parameter */
b = a + fun(a);
```

12

Functional Side Effects

- Two possible solutions to the problem
 1. Write the language definition to disallow functional side effects
 - No two-way parameters in functions
 - No non-local references in functions
 - Advantage: it works!
 - Disadvantage: inflexibility of one-way parameters and lack of non-local references
 2. Write the language definition to demand that operand evaluation order be fixed
 - Disadvantage: limits some compiler optimizations
 - Java requires that operands appear to be evaluated in left-to-right order

13

Overloaded Operators

- Use of an operator for more than one purpose is called operator overloading
- Some are common (e.g., + for `int` and `float`)
- Some are potential trouble (e.g., * in C and C++)
 - Loss of compiler error detection (omission of an operand should be a detectable error)
 - Some loss of readability
 - Can be avoided by introduction of new symbols (e.g., Pascal's `div` for integer division)

14

Overloaded Operators (continued)

- C++, Ada, Fortran 95, and C# allow user-defined overloaded operators
- Potential problems:
 - Users can define nonsense operations
 - Readability may suffer, even when the operators make sense

15

Type Conversions

- A narrowing conversion is one that converts an object to a type that cannot include all of the values of the original type e.g., `float to int`
- A widening conversion is one in which an object is converted to a type that can include at least approximations to all of the values of the original type e.g., `int to float`

16

Type Conversions: Mixed Mode

- A mixed-mode expression is one that has operands of different types
- A coercion is an implicit type conversion
- Disadvantage of coercions:
 - They decrease in the type error detection ability of the compiler
- In most languages, all numeric types are coerced in expressions, using widening conversions
- In Ada, there are virtually no coercions in expressions

17

Explicit Type Conversions

- Called casting in C-based languages
- Examples
 - C: `(int)angle`
 - Ada: `Float (Sum)`

Note that Ada's syntax is similar to that of function calls

18

Type Conversions: Errors in Expressions

- Causes
 - Inherent limitations of arithmetic
e.g., division by zero
 - Limitations of computer arithmetic
e.g. overflow
- Often ignored by the run-time system

19

Relational and Boolean Expressions

- Relational Expressions
 - Use relational operators and operands of various types
 - Evaluate to some Boolean representation
 - Operator symbols used vary somewhat among languages (`!=`, `/=`, `.NE.`, `<>`, `#`)
- JavaScript and PHP have two additional relational operator, `===` and `!==`
 - Similar to their cousins, `==` and `!=`, except that they do not coerce their operands

20

Relational and Boolean Expressions

- Boolean Expressions
 - Operands are Boolean and the result is Boolean
 - Example operators

FORTRAN 77	FORTRAN 90	C	Ada
<code>.AND.</code>	<code>and</code>	<code>&&</code>	<code>and</code>
<code>.OR.</code>	<code>or</code>	<code> </code>	<code>or</code>
<code>.NOT.</code>	<code>not</code>	<code>!</code>	<code>not</code> <code>xor</code>

21

Relational and Boolean Expressions: No Boolean Type in C

- C89 has no Boolean type--it uses `int` type with 0 for false and nonzero for true
- One odd characteristic of C's expressions: `a < b < c` is a legal expression, but the result is not what you might expect:
 - Left operator is evaluated, producing 0 or 1
 - The evaluation result is then compared with the third operand (i.e., `c`)

22

Short Circuit Evaluation

- An expression in which the result is determined without evaluating all of the operands and/or operators
- Example: `(13*a) * (b/13-1)`
If `a` is zero, there is no need to evaluate `(b/13-1)`
- Problem with non-short-circuit evaluation

```
index = 1;
while (index <= length) && (LIST[index] != value)
    index++;
```

 - When `index=length`, `LIST [index]` will cause an indexing problem (assuming `LIST` has `length-1` elements)

23

Short Circuit Evaluation (continued)

- C, C++, and Java: use short-circuit evaluation for the usual Boolean operators (`&&` and `||`), but also provide bitwise Boolean operators that are not short circuit (`&` and `|`)
- Ada: programmer can specify either (short-circuit is specified with `and then` and `or else`)
- Short-circuit evaluation exposes the potential problem of side effects in expressions
e.g. `(a > b) || (b++ / 3)`

24

Assignment Statements

- The general syntax
`<target_var> <assign_operator> <expression>`
- The assignment operator
 - = FORTRAN, BASIC, the C-based languages
 - := ALGOLs, Pascal, Ada
- = can be bad when it is overloaded for the relational operator for equality (that's why the C-based languages use == as the relational operator)

25

Assignment Statements: Conditional Targets

- Conditional targets (Perl)
`($flag ? $total : $subtotal) = 0`

Which is equivalent to

```
if ($flag){
    $total = 0
} else {
    $subtotal = 0
}
```

26

Assignment Statements: Compound Operators

- A shorthand method of specifying a commonly needed form of assignment
- Introduced in ALGOL; adopted by C
- Example

```
a = a + b
```

is written as

```
a += b
```

27

Assignment Statements: Unary Assignment Operators

- Unary assignment operators in C-based languages combine increment and decrement operations with assignment
- Examples

```
sum = ++count (count incremented, added to sum)
```

```
sum = count++ (count incremented, added to sum)
```

```
count++ (count incremented)
```

```
-count++ (count incremented then negated)
```

28

Assignment as an Expression

- In C, C++, and Java, the assignment statement produces a result and can be used as operands
- An example:

```
while ((ch = getchar()) != EOF) {...}
```

`ch = getchar()` is carried out; the result (assigned to `ch`) is used as a conditional value for the `while` statement

29

List Assignments

- Perl and Ruby support list assignments
e.g.,

```
($first, $second, $third) = (20, 30, 40);
```

30

Mixed-Mode Assignment

- Assignment statements can also be mixed-mode, for example

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

- In Fortran, C, and C++, any numeric type value can be assigned to any numeric type variable
- In Java, only widening assignment coercions are done
- In Ada, there is no assignment coercion

31

Summary

- Expressions
- Operator precedence and associativity
- Operator overloading
- Mixed-type expressions
- Various forms of assignment

32