CMPS 3500

Programming Languages

Dr. Chengwei Lei
CEECS
California State University, Bakersfield
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
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 ML and F#, there are no coercions in expressions
Explicit Type Conversions

- Called casting in C-based languages
- Examples
  - C: `(int)angle`
  - F#: `float(sum)`

Note that F#'s syntax is similar to that of function calls
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
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
  - Ruby uses == for equality relation operator that uses coercions and eql? for those that do not
Relational and Boolean Expressions

- Boolean Expressions
  - Operands are Boolean and the result is Boolean
  - Example operators
- C89 has no Boolean type—it uses `int` type with 0 for false and nonzero for true
- One odd characteristic of C’s expressions: \( a < b < c \)
Relational and Boolean Expressions

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

- 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., \( c \))
Short Circuit Evaluation

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

```c
index = 0;
while (index <= length) && (LIST[index] != value)
    index++;
```

- When \(index=\text{length}\), \(LIST[\text{index}]\) will cause an indexing problem (assuming \(LIST\) is \(\text{length} - 1\) long)
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 `|`)

All logic operators in Ruby, Perl, ML, F#, and Python are short-circuit evaluated

Short-circuit evaluation exposes the potential problem of side effects in expressions

e.g. `(a > b) || (b++ / 3)`
Assignment Statements

- The general syntax
  \[ <target\_var> \ <assign\_operator> \ <expression> \]

- The assignment operator
  - Fortran, BASIC, the C-based languages: \( = \)
  - 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)
Assignment Statements: Conditional Targets

- Conditional targets (Perl)

\[(\$flag \ ? \ \$total \ : \ \$subtotal) = 0\]

Which is equivalent to

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

Assignment Statements: Compound Assignment Operators

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

\[
a = a + b
\]

can be written as

\[
a += b
\]
Assignment Statements: Unary Assignment Operators

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

Examples

- `sum = ++count` (count incremented, then assigned to `sum`)
- `sum = count++` (count assigned to `sum`, then incremented)
- `count++` (count incremented)
- `-count++` (count incremented then negated)
Assignment as an Expression

- In the C-based languages, Perl, and JavaScript, the assignment statement produces a result and can be used as an operand

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

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

- Disadvantage: another kind of expression side effect
Multiple Assignments

- Perl, Ruby, and Lua allow multiple-target multiple-source assignments
  
  \[(\text{first}, \text{second}, \text{third}) = (20, 30, 40);\]

  Also, the following is legal and performs an interchange:

  \[(\text{first}, \text{second}) = (\text{second}, \text{first});\]
Assignment in Functional Languages

- Identifiers in functional languages are only names of values
- ML
  - Names are bound to values with `val`
    ```
    val fruit = apples + oranges;
    ```
  - If another `val` for fruit follows, it is a new and different name
- F#
  - F#'s `let` is like ML's `val`, except `let` also creates a new scope
Mixed-Mode Assignment

- Assignment statements can also be mixed-mode
- In Fortran, C, Perl, and C++, any numeric type value can be assigned to any numeric type variable
- In Java and C#, only widening assignment coercions are done
- In Ada, there is no assignment coercion
Summary

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