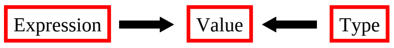
Expressions in Programming Languages

A Programming Language – Universal: All Solvable Computations

- integer values and arithmetic operators (arithmetic expressions)
- variables
- assignment statement
- selection statement
- loop statement/go to statement

Expressions

- An expression is
  - To be evaluated to yield a value of a type.
  - To compute a new value from an old value.



Expressions

- An expression:
  - Operators, operands, parentheses and function calls.
- Expressions:
  - Literals
  - Aggregates
  - Constant and variable access
  - Function calls
  - Arithmetic expression
  - Relational expression
  - Boolean expression
  - Conditional expression

1. Arithmetic Expressions – Design Issues

- What are the operator precedence rules?
- What are the operator associativity rules?
- What is the order of operand evaluation?
- Are there restrictions on operand evaluation side effects?
- Does the language allow user-defined operator overloading?
- What mode mixing is allowed in expressions?

Operator Precedence

- The operator precedence rule:
  - Defines the order in which “adjacent” operators are evaluated.
  - Highest
  - Lowest

```
a + b * c
```

- See p. 295.

Operator Associativity

- The operator associativity rule:
  - Defines the order in which adjacent operators with the same precedence level are evaluated.
  - Left to right
  - Right-to-left

```
a – b + c – d
```

- See p. 297.

CS216 7

Operator: Precedence and Associativity

- Precedence and associativity rules can be overridden with parentheses.

```
(a + b) * c
```

```
a * b + c
```

CS216 8

Operand Evaluation Order

- Irrelevant if neither of the operands of an operator has side effects.
- Crucial when the evaluation of an operand has side effects!
- A side effect of a function call
  - When a function changes either a two-way parameter or a nonlocal variable.

CS216 9

Example: Functions with Side Effects

```
int a = 5;
int fun1() {
   a = 17;
   return 3;
}
void fun2(){
   a = a + fun1();
}

void main() {
   fun2();
}
```

Left-to-right: 8

Right-to-left: 20

CS216 10

Possible Solution 1

- Write the language definition to disallow functional side effects:
  - No two-way parameters in functions
  - No nonlocal references in functions
- Advantage:
  - It works!
- Disadvantage:
  - Programmers want the flexibility.

CS216 11

Possible Solution2

- Write the language definition to demand that the operand evaluation order be fixed.
- Disadvantage:
  - Limits some compiler optimizations

CS216 12

Conditional Expressions

- **`Exp1 ? Exp2 : Exp3`**
  - C, C++, and Java:

```
if (count = 0) then average := 0
else average := sum/count;

average = (count == 0)? 0 : sum / count;
```

CS216 13

Operator Overloading

- Multiple use of the same operator name.
- Advantage:
  - Flexibility
- Disadvantage:
  - Users can define nonsense operations.
  - Readability may suffer.

CS216 14

Mixed-Mode Expression

- An expression that has operands of different types.
  - Need a type conversion.
- Type conversion:
  - Explicit type conversion
  - Implicit type conversion
    - **Type coercion**

CS216 15

Type Conversion

- A narrowing conversion
  - Converted to a type that cannot include all of the values of the original type.
- A widening conversion
  - Converted to a type that can include at least approximations to all of the values of the original type.

CS216 16

Type Coercion – Implicit Type Conversion

- Disadvantages:
  - 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 Modula-2 and Ada, there are virtually no coercions in expressions.

CS216 17

Explicit Type Conversion - Casts

- Doing type conversions explicitly – widening or narrowing.
  - Ada:
    ```
    FLOAT(INDEX)  -- INDEX is INTEGER type
    ```
  - C:
    ```
    (int) speed    /* speed is float type */
    ```

CS216 18

Errors in Expressions

- Caused by:
  - Type mismatch
  - Inherent limitations of arithmetic
    - division by zero
  - Limitations of computer arithmetic
    - overflow, underflow

2. Relational Expressions

- Use relational operators and operands of various types.
- Evaluate to some boolean value.
- See p. 306.

3. Boolean Expressions

- Operands are boolean and the result is boolean value.
- See p. 307.

Short-Circuit Evaluation

- The result (value) of an expression is determined without evaluating all of the operands and/or operators.

```
(a >= 0 ) and (b < 10)
```

Example: Short-Circuit Evaluation

```
list[1..listlen]

index := 1;
while (index <= length) and
              (list[index] <> value) do
  index := index + 1
```

?

Short-Circuit Evaluation

- Pascal:
  - No short-circuit evaluation
- C, C++, and Java:
  - Use short-circuit evaluation for the usual Boolean operators (&& and ||)
- Ada:
  - Programmer can specify either (short-circuit is specified with **`and then`** and **`or else`**)
- FORTRAN 77:
  - Use short-circuit evaluation