

Programming Languages

Tevfik Koşar

Lecture - XI
February 21st, 2006

Roadmap

- Control Flow
 - Basic Paradigms
- Expression Evaluation



Control Flow or Ordering

- Control flow/ordering is fundamental in most models of computing
- Basic paradigms for control flow/ordering:
 - **Sequencing**: the order in which statements appear in code
 - **Selection**: making choices depending on a run-time condition
 - **Iteration**: executing a fragment of code repeatedly
 - **Procedural abstraction**: subroutines, functions
 - **Recursion**: Defining an expression in terms of itself
 - **Concurrency**: Execution of two or more program fragments at the same time
 - **Nondeterminacy**: the ordering is unspecified

3

Expression Evaluation

- **Different styles**:
 - Infix: $(a + b) * c$
 - Prefix: $(* (+ a b) c)$ --eg. Lisp
- In Ada, $a+b$ is short for $+(a,b)$;
- In C++, $a+b$ is short for $a.operator+(b)$;
- **Precedence, associativity** (see Figure 6.1)
 - C has 15 levels - too many to remember
 - Pascal has 4 levels - too few for good semantics
 - Fortran has 8
 - Ada has 6
 - Ada puts *and* & *or* at same level
 - **Lesson**: when unsure, use parentheses!

4

Expression Evaluation

Fortran	Pascal	C	Ada
		++, -- (post-inc., dec.)	
**	not	++, -- (pre-inc., dec.), +, - (unary), &, * (address, contents of), !, ~ (logical, bit-wise not)	abs (absolute value), not, **
*, /	*, /, div, mod, and	* (binary), /, % (modulo division)	*, /, mod, rem
+, - (unary and binary)	+, - (unary and binary), or	+, - (binary)	+, - (unary)
		<<, >> (left and right bit shift)	+, - (binary), & (concatenation)
.eq., .ne., .lt., .le., .gt., .ge. (comparisons)	<, <=, >, >=, =, <>, IN	<, <=, >, >=, (inequality tests)	=, /=, <, <=, >, >=
.not.		==, != (equality tests)	
		& (bit-wise and)	
		^ (bit-wise exclusive or)	
		(bit-wise inclusive or)	
.and.		&& (logical and)	and, or, xor (logical operators)
.or.		(logical or)	
.eqv., .neqv. (logical comparisons)		?: (if...then...else)	
		=, +=, -=, **=, /=, %= >>=, <<=, &=, ^=, = (assignment)	
		, (sequencing)	

Figure 6.1: Operator precedence levels in Fortran, Pascal, C, and Ada. The operators at the top of the figure group most tightly.

5

Expression Evaluation

Precedence:

- If $A < B$ and $C < D$ then..
Causes an error in Pascal since it is grouped as:
If $A < (B \text{ and } C) < D$ then..

Associativity:

- Basic arithmetic expressions almost always associate from left to right
 - $9-3-2$ is 4 not 8
- In Fortran, exponentiation operator associates from right to left
 - $2^{**}3^{**}2$ is 512 ($2^{**}9$) not 64 ($8^{**}2$)
- In Ada exponentiation does not associate
 - You have to write $(2^{**}3)^{**}2$ or $2^{**}(3^{**}2)$
- In C, assignment associates from right to left
 - $a=b=a+c;$

6

Expression Evaluation

- **Short-circuiting**

- Consider `(a < b) && (b < c)`:

- If `a >= b` there is no point evaluating whether `b < c` because `(a < b) && (b < c)` is automatically false

- Other similar situations

```
if (b != 0 && a/b == c) ...
```

```
if (*p && p->foo) ...
```

```
if (f || messy()) ...
```

7

Expression Evaluation

- **Orthogonality**

- Features that can be used in any combination

- Meaning is consistent

```
if (if b != 0 then a/b == c else false) then ...
```

```
if (if f then true else messy()) then ...
```

- First example: Algol 68

```
a:= begin f(b); g(c) end;
```

- In C:

- If `(a = b())` ...

- If `(a == b = c())` ...

8

Expression Evaluation

Combination of Assignment Operators:

In C:

- `a = a + 1;` \rightarrow `a++;` or `++a;`
- `a = a + b;` \rightarrow `a += b;`
- `A[--i] = b;` `A[i--] = b;` difference??

`t1=p; p = p+1; t2=q; q = q+1; *t1 = *t2;`
 \rightarrow `*p++ = *q++;`

9

Expression Evaluation

In languages like Clu, ML, Perl, Python, it is possible..

- **Multiway Assignment:**
 - `a,b := c, d` \rightarrow `a:=c; b:=d;`
 - `a,b := b,a;` \rightarrow `temp:=a; a:=b; b:=temp;`

10

Expression Evaluation

- Assignment
 - statement (or expression) executed for its side effect
 - assignment operators ($+=$, $-=$, etc)
 - handy
 - avoid redundant work (or need for optimization)
 - perform side effects exactly once
 - C $--$, $++$
 - postfix form

11

Expression Evaluation

Applying Mathematical Identities:

$a = b + c;$

$d = c + e + b;$

Will be rearranged by some compilers as:

$a = b + c;$

$a = b + c + e;$

Then, code is generated for:

$a = b + c;$

$d = a + e;$

12

Flow

- Unstructured Flow
 - Go To
- Structured Flow
 - While loop
 - For loop

13

Sequencing

- Sequencing
 - specifies a linear ordering on statements
 - one statement follows another
 - very imperative, Von-Neuman

14

Selection

- Selection

- sequential if statements

```
if ... then ... else
if ... then ... elsif ... else
```

- In Lisp: (cond
 - (C1) (E1)
 - (C2) (E2)
 - ...
 -)

- Case/switch statements (eg. Pascal/C)

- Case ...
 - 1: clause_A;
 - 2,3: clause_B;
 - End;

15

Selection

- Code generated w/o short-circuiting (Pascal)

```
    r1 := A           -- load
    r2 := B
    r1 := r1 > r2
    r2 := C
    r3 := D
    r2 := r2 > r3
    r1 := r1 & r2
    r2 := E
    r3 := F
    r2 := r2 $<>$ r3
    r1 := r1 $|$ r2
    if r1 = 0 goto L2
L1:  then_clause     -- label not actually used
    goto L3
L2:  else_clause
L3:
```

16

Selection

- Code generated w/ short-circuiting (C)

```
    r1 := A
    r2 := B
    if r1 <= r2 goto L4
    r1 := C
    r2 := D
    if r1 > r2 goto L1
L4:   r1 := E
      r2 := F
      if r1 = r2 goto L2
L1:   then_clause
      goto L3
L2:   else_clause
L3:
```