Chapter 6:: Control Flow

Programming Language Pragmatics

Michael L. Scott

Administrative Notes

- Mid-Term Test
  - Thursday, July 27 2006 at 11:30am
  - No lecture before or after the mid-term test
  - You are responsible for material presented in the lectures not necessarily covered in the textbook
  - Will cover chapters1-6
    - Chapter 1: complete
    - Chapter 2: chapter introduction and section 2.1
    - Chapter 3: entire chapter except Section 3.5.1 & 3.5.2
    - Chapter 5: entire chapter
    - Chapter 6:

Administrative Notes

- Assignment One
  - Due date: Friday, July 28 2006 at 1:00pm
  - Submit your assignment in the drop-box located at the Computer Science and Engineering undergraduate office
  - Late assignments are subject to a penalty of 10% each day
  - I may choose to mark only a subset of the assigned questions
  - You must "show your work" where appropriate to obtain full marks

Administrative Details

- Drop Deadline
  - Computer Science Department drop-deadline for the course is July 31 2006
  - If you wish to drop the course before this deadline will have to petition to "drop late" as far as the registrars office is concerned, the drop deadline has passed (they treated the course as D2 - see registrars office)
    - Of course, the petition will have the support of the department and of myself as well and will be approved under these circumstances

Review

- Describe the computer’s memory hierarchy
- What are registers and what are their significance ?
- What is the problem with accessing data in main memory ?
- What is RISC and what is CISC ?
- What are the advantages/disadvantages of RISC and CISC ?
- What is little endian/big endian and what are the advantages/disadvantages of each ?

Review

- What is an addressing mode ?
- Describe several different addressing modes
- What is a load delay ?
- What is a branch delay ?
- What is microprogramming ?
- What can cause a pipeline to stall ?
- What is a instruction scheduling ?
- What are the dependences amongst instructions associated with instruction scheduling ?

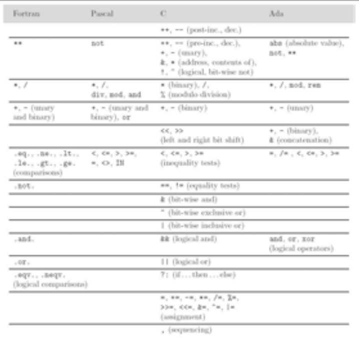
Control Flow

- Basic paradigms for control flow:
  - Sequencing
  - Selection
  - Iteration
  - Subroutines, recursion (and related control abstractions, e.g. iterators)
  - Nondeterminacy
  - Concurrency

Expression Evaluation

- Infix, prefix operators
- Precedence, associativity (see Figure 6.1)
  - C has 15 levels - too many to remember
  - Pascal has 3 levels - too few for good semantics
  - Fortran has 8
  - Ada has 6
    - Ada puts *and* & *or* at same level
  - **Lesson**: when unsure, use parentheses!

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.

Expression Evaluation

- Ordering of operand evaluation (generally none)
- Application of arithmetic identities
  - distinguish between commutativity, and (assumed to be safe)
  - associativity (known to be dangerous)

    `(a + b) + c` works if `a~=maxint` and `b~=minint` and `c<0`

    `a + (b + c)` does not
  - inviolability of parentheses

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()) ...
```

Expression Evaluation

- Variables as values vs. variables as references
  - value-oriented languages
    - C, Pascal, Ada
  - reference-oriented languages
    - most functional languages (Lisp, Scheme, ML)
    - Clu, Smalltalk
  - Algol-68 kinda halfway in-between
  - Java deliberately in-between
    - built-in types are values
    - user-defined types are objects - references

Expression Evaluation

- Expression-oriented vs. statement-oriented languages
  - expression-oriented:
    - functional languages (Lisp, Scheme, ML)
    - Algol-68
  - statement-oriented:
    - most imperative languages
  - C kinda halfway in-between (distinguishes)
    - allows expression to appear instead of statement

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 ...
```

- Initialization
  - Pascal has no initialization facility (assign)
- Aggregates
  - Compile-time constant values of user-defined composite types

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

Expression Evaluation

- Side Effects
  - often discussed in the context of functions
  - a side effect is some permanent state change caused by execution of function
    - some noticable effect of call other than return value
    - in a more general sense, assignment statements provide the ultimate example of side effects
      - they change the value of a variable

Expression Evaluation

- SIDE EFFECTS ARE FUNDAMENTAL TO THE WHOLE VON NEUMANN MODEL OF COMPUTING

- In (pure) functional, logic, and dataflow languages, there are no such changes
  - These languages are called SINGLE-ASSIGNMENT languages

Expression Evaluation

- Several languages outlaw side effects for functions
  - easier to prove things about programs
  - closer to Mathematical intuition
  - easier to optimize
  - (often) easier to understand
- But side effects can be nice
  - consider rand()

Expression Evaluation

- Side effects are a particular problem if they affect state used in other parts of the expression in which a function call appears
  - It's nice not to specify an order, because it makes it easier to optimize
  - Fortran says it's OK to have side effects
    - they aren't allowed to change other parts of the expression containing the function call
    - Unfortunately, compilers can't check this completely, and most don't at all

Sequencing

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

Selection

- Selection
  - sequential if statements

```
if ... then ... else
if ... then ... elsif ... else
(cond
      (C1)  (E1)
      (C2)  (E2)
      ...
      (Cn)  (En)
      (T)   (Et)
)
```