Chapter 6:: Control Flow (cont.)

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

Administrative Notes

- Mid-Term Test (cont.)
  - 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, 3.3.4 and 3.3.5
    - Chapter 5: entire chapter
    - Chapter 6: chapter introduction & Section 6.1 (complete)

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

- What is an expression ?
- What is a statement ?
- What are the seven control flow paradigms ?
- What are prefix, postfix operators ?
- What are precedence rules and why are the important ?
- What is the only way to ensure precedence results in exactly what we “meant” ?
- In which language do all operations have the same precedence ?

Review

- What is short-circuiting ?
- What are the benefits of short-circuiting ?
- It is orthogonality and what are its benefits ?
- Describe the difference between expression oriented and statement oriented languages
- What is a side-effect ?
- Do imperative languages allow side-effects ?
- Is associtivity safe ? How about commutatitvity ?

Control Flow

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

Selection

- Selection
  - sequential if statements

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


Selection

- Selection
  - Fortran computed gotos
  - jump code
    - for selection and logically-controlled loops
    - no point in computing a Boolean value into a register, then testing it
    - instead of passing register containing Boolean out of expression as a synthesized attribute, pass inherited attributes INTO expression indicating where to jump to if true, and where to jump to if false

Selection

- Jump is especially useful in the presence of short-circuiting
- **Example** (section 6.4.1 of book):

```
if ((A > B) and (C > D)) or (E <> F)
  then
   then_clause
  else
   else_clause
```


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:
```


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:
```


Iteration

- Enumeration-controlled
  - Pascal or Fortran-style for loops
    - scope of control variable
    - changes to bounds within loop
    - changes to loop variable within loop
    - value after the loop

Iteration

- The `goto` controversy
  - *assertion*: `gotos` are needed almost exclusively to cope with lack of one-and-a-half loops
  - early return from procedure
  - exceptions
  - in many years of programming, I can't remember using one for any other purpose
    - except maybe complicated conditions that can be separated into a single if-then-else because of the need for short-circuiting

Recursion

- Recursion
  - equally powerful to iteration
  - mechanical transformations back and forth
  - often more intuitive (sometimes less)
  - *naïve* implementation less efficient
    - no special syntax required
    - fundamental to functional languages like Scheme

Recursion

- Tail recursion
  - No computation follows recursive call

```
        /* assume a, b > 0 */
int gcd (int a, int b) {
   if (a == b) return a;
   else if (a > b) return gcd (a - b, b);
   else return gcd (a, b - a);
}
```