Ordering – Control Flow

Intermediate Representations

- Structural (graphically oriented)
  - E.g. abstract syntax tree, directed acyclic graph
- Linear
  - E.g., stack machine/one-address code, 3-address code
- Hybrids
  - E.g., control flow graph

Basic Paradigms for Control Flow

- Sequencing
- Selection
- Iteration
- Procedural abstraction
- Recursion
- Non-determinacy
- concurrency

Expression Evaluation

- Precedence
  - Among arithmetic, relational (comparison), and logical operators
  - APL and Smalltalk give all operators equal precedence – parentheses required
- Associativity
  - Rules more uniform across languages – left associative
  - E.g., exception – exponentiation in Fortran associates right to left
  - Solution: ALWAYS USE PARENTHESES WHEN UNSURE!

Ordering Within Expressions

- Many languages do not specify an evaluation order for components within an expression (Java provides left to right evaluation)
  - Easier to optimize
- Problem: side effects, in particular, permanent state change caused by execution of a function
- Solutions:
  - Don't allow functions to have side effects
  - Allow them (e.g., Fortran), but don't allow modification of variables in the expression – hard to check

Short-Circuit Evaluation

- Short-circuit evaluation of boolean expressions
  - Avoids extra computation overhead
  - Semantics
    - Presence or absence of dynamic semantic errors
  - If a sub-expression can cause side effects, may not be desirable to short-circuit

Assignments

- Value model versus reference model
- Initialization – what guarantees does the language provide
  - C++ and Java – constructors
  - C – statically allocated uninitialized variables initialized with zero
  - Java – must be "definitely assigned" (assigned on every possible control flow path) prior to use based on control flow
- Initialization versus assignment
  - Assignment requires both allocation and deallocation when performing storage management
- Combination assignment operators – e.g., +=, ++, --

Structured Vs. Unstructured Flow

- The goto controversy
  - Follows underlying assembly/machine code closely
  - Primary uses
    - Mid-loop exit and continue
      - Continue statement in C
    - Early returns from subroutines
      - Explicit return statement (Fortran, Algol descendants)
    - Errors and other exceptions
      - Structured exception handling as in Java
      - Continuations – code address along with referencing environment (e.g., Scheme)

Selection

- If-then-else
- Case statements – introduced in Algol-W
  - Syntactic elegance
  - Generation of efficient target code
    - Jump tables – dense range
    - Search table (hashing, binary search, search tree)
    - Sequential testing (small number of choices, non-dense range)

Iteration

- Enumeration-controlled
  - Can loop index and control variables be modified in the loop and if so, what is the effect on control
  - Is the loop always executed at least once
  - What is the value of the loop index variable on exiting the loop
  - Can control jump into the loop from outside
- Logically controlled loops
  - Pre-test
  - Post-test
  - Mid-test

Recursion

- No special construct required – allow subroutines to call themselves or to call other subroutines that call them back in turn
- Iteration based on repeated modification of variables, recursion requires no side-effects
- Naive implementation of iteration more efficient than recursion

Nondeterminacy

- Guarded commands
  - Non-deterministic choice made among guards that evaluate to true (only one set of statements evaluated)
  - Allows formal reasoning about correctness of code
  - Useful when dealing with concurrency

Subroutines – Calling Sequence

- Prologue
  - Parameter passing
  - Saving return address
  - Changing program coutner
  - Changing stack pointer to allocate frame
  - Saving registers (including frame pointer)
  - Changing frame pointer to refer to new frame
  - Executing initialization code for any objects on new frame
- Epilogue
  - Passing return parameters or function values
  - Executing finalization code
  - Deallocating the stack frame
  - Restoring other saved registers (including frame pointer)
  - Restoring program counter

Parameter Passing

- Formal versus actual parameters
- Parameter passing modes
  - Call by value
  - Call by reference
  - Call by result
  - Call by name
- Closure – reference to a subroutine along with its referencing environment
- Variable numbers of arguments
  - C, C++, Common Lisp