8. Statement-Level Control Structure

8.1 Introduction

- Levels of Control Flow
  - Within expressions
  - Among program units
  - Among program statements
- Evolution
  - FORTRAN I control statements were based directly on IBM 704 hardware
  - Much research and argument in the1960s about the issue
  - One important result: It was proven that all flowcharts can be coded with only two-way selection and pretest logical loops
- A control structure is a control statement and the statements whose execution it controls
- Overall Design Question
  - What control statements should a language have, beyond selection and pretest logical loops?

8.2 Compound statements

- Introduced by ALGOL 60
- A block is a compound statement that can define a new scope (with local variables)

8.3 Selection Statements

- Design Issues
  - What is the form and type of the control expression?
  - What is the selectable segment form (single statement, statement sequence, compound statement)?
  - How should the meaning of nested selectors be specified?
- Single-Way Examples
  - FORTRAN IF: IF (boolean_expr) statement
  - Problem: can select only a single statement; to select more, a `GOTO` must be used, as in the following example

```
    IF (.NOT. condition) GOTO 20
         ...
         ...
 20 CONTINUE
```

  - Negative logic is bad for readability
  - This problem was solved in FORTRAN 77
  - Most later languages allow compounds for the selectable segment of their single-way selectors

8.3 Selection Statements (continued)

- Two-way Selector Examples
- ALGOL 60 `if`:

```
if (boolean_expr)
   then statement (the then clause)
   else statement (the else clause)
```

  - The statements could be single or compound
- Nested Selectors
  - e.g. (Java)

```
if ...
    if ...
      ...
  else ...
```

  - Which if gets the else?
  - Java's static semantics rule: `else` goes with the nearest `if`

8.3 Selection Statements (continued)

- ALGOL 60's solution - disallow direct nesting

```
if ... then
  begin
if ...
  then ...
  else ...
end
```

```
if ... then
  begin
  if ... then ...
  end
else ...
```

- FORTRAN 90 and Ada solution - closing special words

e.g. (Ada)

```
if ... then
   if ... then
     ...
   else
     ...
   end if
 end if
```

```
if ... then
   if ... then
     ...
   end if
else
   ...
end if
```

- Advantage: flexibility and readability

8.3 Selection Statements (continued)

- Multiple Selection Constructs
- Design Issues
  - What is the form and type of the control expression?
  - What segments are selectable (single, compound, sequences)?
  - Is the entire construct encapsulated?
  - Is execution flow through the structure restricted to include just a single selectable segment?
  - What is done about unrepresented expression values?
- Early Multiple Selectors
  - FORTRAN arithmetic `IF` (a three-way selector) `IF` (arithmetic expression) N1, N2, N3
  - Bad aspects
    - Not encapsulated (selectable segments could be anywhere)
    - Segments require `GOTO`s
  - FORTRAN computed `GOTO` and assigned `GOTO`

8.3 Selection Statements (continued)

- Modern Multiple Selectors
- Pascal `case` (from Hoare's contribution to ALGOL W)

```
case expression of
   constant_list_1 : statement_1;
   ...
   constant_list_n : statement_n
end
```

  - Design choices
    - Expression is any ordinal type (`int`, `boolean`, `char`, `enum`)
    - Segments can be single or compound
    - Construct is encapsulated
    - Only one segment can be executed per execution of the construct
    - In Wirth's Pascal, result of an unrepresented control expression value is undefined (In 1984 ISO Standard, it is a runtime error)
  - Many dialects now have `otherwise` or `else` clause

8.3 Selection Statements (continued)

- The C, C++, and Java switch

```
switch (expression) {
  constant_expression_1 : statement_1;
  ...
  constant_expression_n : statement_n;
    [default: statement_n+1]
}
```

- Design Choices: (for `switch`)
  - Control expression can be only an integer type
  - Selectable segments can be statement sequences, blocks, or compound statements
  - Construct is encapsulated
  - Any number of segments can be executed in one execution of the construct (there is no implicit branch at the end of selectable segments) (a trade-off between reliability and flexibility -- convenience)
    - To avoid it, the programmer must supply a `break` statement for each segment
  - default clause is for unrepresented values (if there is no `default`, the whole statement does nothing)

8.3 Selection Statements (continued)

- Ada's case is similar to Pascal's case, except:
  - Constant lists can include:
  - Subranges e.g., `10..15`
  - Boolean OR operators

    e.g., `1..5 | 7 | 15..20`
  - Lists of constants must be exhaustive
    - Often accomplished with others clause
    - This makes it more reliable
- Multiple Selectors can appear as direct extensions to two-way selectors, using else-if clauses (ALGOL 68, FORTRAN 90, Ada)

Ada:

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

  - Far more readable than deeply nested `if`'s
  - Allows a Boolean gate on every selectable group

8.4 Iterative Statements

- The repeated execution of a statement or compound statement is accomplished either by iteration or recursion; here we look at iteration, because recursion is unit-level control
- General design Issues for iteration control statements
  - How is iteration controlled?
  - Where is the control mechanism in the loop?
- **Counter-Controlled Loops**
- Design Issues
  - What are the type and scope of the loop var?
  - What is the value of the loop var at loop termination?
  - Should it be legal for the loop var or loop parameters to be changed in the loop body, and if so, does the change affect loop control?
  - Should the loop parameters be evaluated only once, or once for every iteration?

8.4 Iterative Statements (continued)

- FORTRAN 90
  - Syntax: `DO` label var = start, finish [, stepsize]
  - Stepsize can be any value but zero
  - Parameters can be expressions
  - Design choices:
    - Loop var must be `INTEGER`
  - Loop var always has its last value
  - The loop var cannot be changed in the loop, but the parameters can; because they are evaluated only once, it does not affect loop control
  - Loop parameters are evaluated only once
- FORTRAN 90's Other `DO`
  - Syntax

```
[name:] DO variable = initial, terminal [, stepsize]
        …
        END DO [name]
```

  - Loop var must be an `INTEGER`

8.4 Iterative Statements (continued)

- ALGOL 60
  - Syntax: `for` var := <list_of_stuff> `do` statement, where <list_of_stuff> can have
    - list of expressions
    - expression `step` expression `until` expression
    - expression `while` boolean_expression

```
for index := 1 step 2 until 50,
                60, 70, 80,
                index + 1 until 100 do
(index = 1, 3, 5, 7, ..., 49, 60, 70, 80, 81, 82,
          ...,100)
```

- ALGOL 60 Design choices
  - Control expression can be `int` or `real`; its scope is whatever it is declared to be
  - Control var has its last assigned value after loop termination
  - The loop var cannot be changed in the loop, but the parameters can, and when they are, it affects loop control
  - Parameters are evaluated with every iteration, making it very complex and difficult to read

8.4 Iterative Statements (continued)

- Pascal
  - Syntax:
  - `for` variable := initial (`to` | `downto`) final `do` statement
  - Design Choices
    - Loop var must be an ordinal type of usual scope
    - After normal termination, loop var is undefined
    - The loop var cannot be changed in the loop; the loop parameters can be changed, but they are evaluated just once, so it does not affect loop control
- Ada
  - Syntax:

```
for var in [reverse] discrete_range loop
  ...
end loop
```

  - Design choices
    - Type of the loop var is that of the discrete range; its scope is the loop body (it is implicitly declared)
    - The loop var does not exist outside the loop
    - The loop var cannot be changed in the loop, but the discrete range can; it does not affect loop control
    - The discrete range is evaluated just once

8.4 Iterative Statements (continued)

- C
  - Syntax

    for ([expr_1] ; [expr_2] ; [expr_3]) statement
  - The expressions can be whole statements, or even statement sequences, with the statements separated by commas
  - The value of a multiple-statement expression is the value of the last statement in the expression e.g.,
  - for (i = 0, j = 10; j == i; i++) …
  - If the second expression is absent, it is an infinite loop
  - Design Choices:
    - There is no explicit loop var
    - Irrelevant
    - Everything can be changed in the loop
    - The first expression is evaluated once, but the other two are evaluated with each iteration
  - This loop statement is the most flexible

8.4 Iterative Statements (continued)

- C++
  - Differs from C in two ways:
    - The control expression can also be Boolean
    - The initial expression can include variable definitions (scope is from the definition to the end of the loop body)
- Java
  - Differs from C++ in that the control expression must be Boolean

8.4 Iterative Statements (continued)

- **Logically-Controlled Loops**
- Design Issues
  - Pretest or postest?
  - Should this be a special case of the counting loop statement (or a separate statement)?
- Language Examples
  - Pascal has separate pretest and posttest logical loop statements (`while-do` and `repeat-until`)
  - C and C++ also have both, but the control expression for the posttest version is treated just like in the pretest case (`while - do` and `do - while`)
  - Java is like C, except the control expression must be Boolean (and the body can only be entered at the beginning -- Java has no goto)
  - Ada has a pretest version, but no posttest
  - FORTRAN 77 and 90 have neither
  - Perl has two pretest logical loops, `while` and `until`, but no posttest logical loop

8.4 Iterative Statements (continued)

- User-Located Loop Control Mechanisms
- Design issues:
  - Should the conditional be part of the exit?
  - Should the mechanism be allowed in an already controlled loop?
  - Should control be transferable out of more than one loop?
- Examples
  - Ada - conditional or unconditional; for any loop; any number of levels

```
for ... loop
    ...
    exit when ...
    ...
  end loop
```

```
LOOP1:
   while ... loop
     ...
LOOP2:
   for ... loop
     ...
     exit LOOP1 when ..
     ...
   end loop LOOP2;
   ...
 end loop LOOP1;
```

8.4 Iterative Statements (continued)

- C , C++, and Java - `break`
  - Unconditional; for any loop or `switch`; one level only (except Java's can have a label)
  - There is also has a `continue` statement for loops; it skips the remainder of this iteration, but does not exit the loop
- FORTRAN 90 - `EXIT`
  - Unconditional; for any loop, any number of levels
  - FORTRAN 90 also has `CYCLE`, which has the same semantics as C's `continue`

- Iteration Based on Data Structures
  - Concept: use order and number of elements of some data structure to control iteration
  - Control mechanism is a call to a function that returns the next element in some chosen order, if there is one; else exit loop
  - C's for can be used to build a user-defined iterator

    e.g. `for (p=hdr; p; p=next(p)) { ... }`
  - Perl has a built-in iterator for arrays and hashes

    e.g., `foreach $name (@names) { print $name }`

8.5 Unconditional Branching

- Problem: readability
  - Some languages do not have them: e.g., Java
- Loop exit statements are restricted and somewhat camouflaged goto's
- Label forms
  - Unsigned int constants: Pascal (with colon) FORTRAN (no colon)
  - Identifiers with colons: ALGOL 60, C
  - Identifiers in << ... >>: Ada
  - Variables as labels: PL/I
    - Can be assigned values and passed as parameters
    - Highly flexible, but make programs impossible to read and difficult to implement

8.6 Guarded Commands

- Dijkstra, 1975
- Purpose: to support a new programming methodology (verification during program development)
- Selection: `if` <boolean> -> <statement>

  [] <boolean> -> <statement>

  ...

  [] <boolean> -> <statement>

  `fi`
  - Semantics: when this construct is reached,
    - Evaluate all boolean expressions
    - If more than one are true, choose one nondeterministically
    - If none are true, it is a runtime error
  - Idea: if the order of evaluation is not important, the program should not specify one
  - See book examples (pp. 339, 340)

8.6 Guarded Commands (continued)

- Loops `do` <boolean> -> <statement>

  [] <boolean> -> <statement>

  ...

  [] <boolean> -> <statement>

  `od`
  - Semantics: For each iteration
    - Evaluate all boolean expressions
    - If more than one are true, choose one nondeterministically; then start loop again
    - If none are true, exit loop
  - --> See book example (p. 340)
- Connection between control statements and program verification is intimate
  - Verification is impossible with gotos
  - Verification is possible with only selection and logical pretest loops
  - Verification is relatively simple with only guarded commands
- Chapter Conclusion: Choice of control statements beyond selection and logical pretest loops is a trade-off between language size and writability