Chapter 8
Statement-Level Control Structures

• Introduction
• Selection Statements
• Multi-way selection stmts
• Iterative Statements
• Unconditional Branching
• Guarded Commands
• Conclusions

Multiple-Way Selection Statements

• Design Issues:
  1. What is the form and type of the control expression?
  2. How are the selectable segments specified?
  3. Is execution flow through the structure restricted to include just a single selectable segment?
  4. What is done about unrepresented expression values?

Multiple-Way Selection Statements

• Early Multiple Selectors:
  1. FORTRAN arithmetic IF (a three-way selector)
     IF (arithmetic expression) L1, L2, L3
• Bad aspects:
  – Not encapsulated (selectable segments could be anywhere)
  – Segments require GOTOs
Multiple-Way Selection Statements

• Modern Multiple Selectors
  1. Pascal case (from Hoare's contribution to ALGOL W)
     ```pascal
case expression of
  constant_list_1 : statement_1;
  ...
  constant_list_n : statement_n
end
```

Multiple-Way Selection Statements

• Design choices of Pascal:
  1. Expression is any ordinal type
     (int, boolean, char, enum)
  2. Segments can be single or compound
  3. Only one segment can be executed per execution of the construct
  4. 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

Multiple-Way Selection Statements

2. The C, C++, and Java switch
  ```java
switch (expression)
{
  constant_expression_1 : statement_1;
  ...
  constant_expression_n : statement_n;
  [default : statement_n+1]
}
```
Multiple-Way Selection Statements

• Design Choices: (for switch in C/C++/Java)
  1. Control expression can be only an integer type
  2. Selectable segments can be statement sequences, blocks, or compound statements
  3. Any number of segments can be executed in one execution of the construct. To avoid it, the programmer must supply a break statement for each segment.
  4. default clause is for unrepresented values (if there is no default, the whole statement does nothing)

Multiple-Way Selection Statements

• 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

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:
  1. How is iteration controlled?
  2. Where is the control mechanism in the loop?
Iterative Statements

1. Counter-Controlled Loops
   • Design Issues:
     1. What are the type and scope of the loop variable?
     2. What is the value of the loop variable at loop termination?
     3. Should it be legal for the loop variable or loop parameters to be changed in the loop body, and if so, does the change affect loop control?
     4. Should the loop parameters be evaluated only once, or once for every iteration?

Iterative Statements

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

Iterative Statements

• FORTRAN 90’s Other DO
  – Syntax:
    [name:] DO variable = initial, terminal [, stepsize]
    
    END DO [name]
  – Loop variable must be an INTEGER
Iterative Statements

2. ALGOL 60
   • Syntax: \texttt{for \ var := \{list\_of\_stuff\} \ do \ statement}
     where \{list\_of\_stuff\} can have:
     - list of expressions
     - expression \texttt{step} expression \texttt{until} expression
     - expression \texttt{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)

Iterative Statements

• ALGOL 60 Design choices:
  1. Control expression can be \texttt{int} or \texttt{real}; its scope
     is whatever it is declared to be
  2. Control variable has its last assigned value after
     loop termination
  3. The loop variable cannot be changed in the loop,
     but the parameters can, and when they are, it
     affects loop control
  4. Parameters are evaluated with every iteration,
     making it very complex and difficult to read

Iterative Statements

3. Pascal
   • Syntax: \texttt{for \ variable := \{initial \| to \| \downto\} \ final \ do \ statement}
   • Design Choices:
     1. Loop variable must be an ordinal type of usual scope
     2. After normal termination, loop variable is undefined
     3. The loop variable 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
     4. Just once
Iterative Statements

4. Ada
• Syntax:
  for var in [reverse] discrete_range loop
  ... end loop

Iterative Statements

• Ada Design choices:
  1. Type of the loop variable is that of the discrete range; its scope is the loop body (it is implicitly declared)
  2. The loop variable does not exist outside the loop
  3. The loop variable cannot be changed in the loop, but the discrete range can; it does not affect loop control
  4. The discrete range is evaluated just once

Iterative Statements

5. 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
Iterative Statements

• C Design Choices:
  1. There is no explicit loop variable
  2. Irrelevant (no explicit loop-var so value after loop is irrelevant)
  3. Everything can be changed in the loop
  4. The first expression is evaluated once, but the other two are evaluated with each iteration
• This loop statement is the most flexible

6. C++
   • Differs from C in two ways:
     1. The control expression can also be Boolean
     2. The initial expression can include variable definitions (scope is from the definition to the end of the loop body)

7. Java
   • Differs from C++ in that the control expression must be Boolean

2. Logically-Controlled Loops
   • Design Issues:
     1. Pretest or posttest?
     2. Should this be a special case of the counting loop statement (or a separate statement)?
Iterative Statements

• Language Examples:
  1. Pascal has separate pretest and posttest logical loop statements (while-do and repeat-until).
  2. 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).
  3. 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).

Iterative Statements

• Language Examples (continued):
  4. Ada has a pretest version, but no posttest.
  5. FORTRAN 77 and 90 have neither.
  6. Perl has two pretest logical loops, while and until, but no posttest logical loop.

Iterative Statements

3. User-Located Loop Control Mechanisms
• Design issues:
  1. Should the conditional be part of the exit?
  2. Should control be transferable out of more than one loop?
Iterative Statements

• Examples:
  1. Ada - conditional or unconditional; for any loop; any number of levels

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

• 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))`
    ```
    { ... }
    ```

2. C, C++, and Java - `break`

  • Unconditional; for any loop or `switch`; one level only (except Java’s can have a label)
  • There is also a `continue` statement for loops; it skips the remainder of this iteration, but does not exit the loop
Iterative Statements

- Perl has a built-in iterator for arrays and hashes
e.g.,
`foreach $name (@names) { print $name }`

Unconditional Branching

- Problem: readability
- Some languages do not have them: e.g., Java
- Loop exit statements are restricted and somewhat camouflaged `goto`'s

Unconditional Branching

- Label forms:
  1. Unsigned int constants: Pascal (with colon)
      FORTRAN (no colon)
  2. Identifiers with colons: ALGOL 60, C
  3. Identifiers in `<< ... >>`: Ada
Guarded Commands

- Dijkstra, 1975
- Purpose: to support a new programming methodology (verification during program development)
- Idea: if the order of evaluation is not important, the program should not specify one

Guarded Commands

1. Selection:
   \[
   \text{if } \langle \text{boolean} \rangle \rightarrow \langle \text{statement} \rangle \\
   \ldots \\
   \text{fi}
   \]

   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

Guarded Commands

2. Loops:
   \[
   \text{do } \langle \text{boolean} \rangle \rightarrow \langle \text{statement} \rangle \\
   \langle \text{boolean} \rangle \rightarrow \langle \text{statement} \rangle \\
   \ldots \\
   \text{od}
   \]

   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
Guarded Commands

- 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

Conclusion

- Choice of control statements beyond selection and logical pretest loops is a trade-off between language size and writability