Control Structures

In Text: Chapter 7

Outline

- Control structures
- Selection
  - One-way
  - Two-way
  - Multi-way
- Iteration
  - Counter-controlled
  - Logically-controlled
- Gotos
- Guarded statements

Levels of Control Flow

- Within expressions
- Among program statements
- Among program units

Evolution of Control Structures

- 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

Control Structures

- 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?
- Single entry/single exit are highly desirable (a lesson learned from structured programming)

Compound Statements

- Introduced by ALGOL 60 in the form of begin...end
- A block is a compound statement that can define a new scope (with local variables)

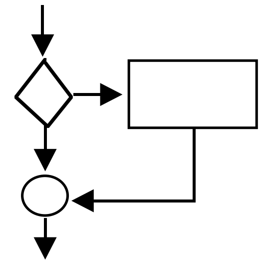
Selection Statements

- 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 Selection

- One-way "if" statement
- FORTRAN IF:

IF (boolean_expr) statement

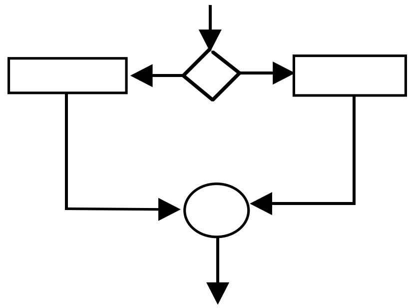


- Problem: can select only a single statement; to select more, a goto must be used

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

Two-Way Selection

- "if-then-else" statement
- ALGOL 60 if:



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

- The statements could be single or compound

Nested Selectors

- Pascal:

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

- Which “then” gets the “else”?
- Pascal's rule: else goes with the nearest then

Disallowing Direct Nesting

- ALGOL 60's solution—disallows direct nesting

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

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

Closing Reserved Words

- FORTRAN 77, Ada, Modula-2 solution—closing special words
- In Ada:

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

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

- Advantage: flexibility and readability
- Modula-2 uses END for all control structures
  - This results in poor readability

Multiple Selection Constructs

- Design Issues:
  - What is the form and type of the control expression?
  - What segments are selectable (single, compound, sequential)?
  - Is the entire construct encapsulated?
  - Is execution flow through the structure restricted to include just a single selectable segment?
  - What about unrepresented expression values?

Early Multiple Selectors:

- FORTRAN arithmetic IF (a three-way selector)

IF (arithmetic expression) N1, N2, N3

- Disadvantages:
  - Not encapsulated (selectable segments could be anywhere)
  - Segments require GOTOs
- FORTRAN computed GOTO and assigned GOTO

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
```

Case: Pascal 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

C/C++ 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 or blocks
  - Construct is encapsulated
  - Any number of segments can be executed in one execution of the construct (reliability vs. flexibility)
  - Default clause for unrepresented values

Case: Ada Design Choices

```
case expression is
    when constant_list_1 => statement_1;
    ...
    when constant_list_n => statement_n;
endSimilar to Pascal's case, except:
```

- Similar to Pascal
- Constant lists can include:
  - Subranges: 10..15
  - Multiple choices: 1..5 | 7 | 15..20
- Lists of constants must be exhaustive (more reliable)
- Often accomplished with others clause

Multi-Way If Statements

- Multiple Selectors can appear as direct extensions to two-way selectors, using else-if clauses (ALGOL 68, FORTRAN 77, Modula-2, 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

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?
- Two common strategies: counter-controlled, and logically-controlled

Counter-Controlled Loops

- Design Issues:
- What is the type and scope of the loop variable?
- What is the value of the loop variable at loop termination?
- Should it be legal for the loop variable or loop parameters to be changed in the loop body?
- If so, does the change affect loop control?
- Should the loop parameters be evaluated only once, or once for every iteration?

FORTRAN DO Loops

- FORTRAN 77 and 90
- Syntax:

DO label var = start, finish [, stepsize]

- Stepsize can be any value but zero
- Parameters can be expressions
- Design choices:
  - Loop var can be INTEGER, REAL, or DOUBLE
  - Loop var always has its last value
  - Loop parameters are evaluated only once
  - 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

FORTRAN 90's Other DO

- Syntax:

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

- Loop var must be an INTEGER

ALGOL 60 For Loop

- 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 For 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

Pascal For Loop

- Syntax:

```
for var := 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 For Loop

- 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

C For Loop

- 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
- If the second expression is absent, it is an infinite loop

C For Loop Design Choices

- There is no explicit loop variable
- Everything can be changed in the loop
- Pretest
- The first expression is evaluated once, but the other two are evaluated with each iteration
- This loop statement is the most flexible

C++ & Java For Loops

- 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 body of the loop
- Java is the same, except the control expression must be Boolean

Logically-Controlled Loops

- Design Issues:
  - Pretest or post-test?
  - Should this be a special case of the counting loop statement, or a separate statement?

Logic Loops: Examples

- Pascal: separate pretest and posttest logical loop statements (while-do and repeat-until)
- C and C++: also have both, but the control expression for the post-test version is treated just like in the pretest case (while - do and do - while)
- Java: like C, except the control expression must be Boolean (and the body can only be entered at the beginning—Java has no goto)
- Ada: a pretest version, but no post-test
- FORTRAN 77 and 90: have neither

User-Located Loop Controls

- Statements like break or continue
- Design issues:
  - Should the conditional be part of the exit?
  - Should the mechanism be allowed in logically- or counter-controlled loops?
  - Should control be transferable out of more than one loop?

User-Located Controls: Ada

- Can be 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;
```

User-Loc. Controls: More Examples

- C, C++, Java:
  - Break: unconditional; for any loop or switch; one level only (except Java)
  - Continue: skips the remainder of this iteration, but does not exit the loop
- FORTRAN 90:
  - EXIT: Unconditional; for any loop, any number of levels
  - CYCLE: same as C's continue

Iteration Based on Data Structures

- Concept: use order and number of elements of some data structure to control iteration
- Two strategies:
  - "Passive" iterator: provide a set of functions for a data structure that the user can use to construct a loop using while, for, etc.
  - "Active" iterator: encapsulate the loop control in an operation, and only allow the user to provide the loop body; in other words, provide a "functional form" or a template operation for the entire loop

Unconditional Branching (GOTO)

- Problem: readability
- Some languages do not have them: e.g., Modula-2 and Java
- They require some kind of statement label
- Label forms:
  - Unsigned int constants: Pascal (with colon), FORTRAN (no colon)
  - Identifiers with colons: ALGOL 60, C, C++
  - Identifiers in << ... >>: Ada

Variables as labels: PL/I

- Can store a label value in a variable
- Can be assigned values and passed as parameters
- Highly flexible, but make programs impossible to read and difficult to implement

Restrictions on Pascal's Gotos

- A statement group is either a compound statement or the body of a repeat-until
- The target of a goto cannot be a statement in a statement group that is not active
- Means the target can never be in a statement group that is at the same level or is nested more deeply than the one with the goto
- An important remaining problem: the target can be in any enclosing subprogram scope, as long as the statement is not in a statement group
- This means that a goto can terminate any number of subprograms

Guarded Commands (Dijkstra, 1975)

- Purpose: to support a new programming methodology (verification during program development)
- Also useful for concurrency
- Two guarded forms:
  - Selection (guarded if)
  - Iteration (guarded while)

Guarded 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 (p. 319)

Guarded Iteration

```
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. 320)

Choice of Control Statements

- Beyond selection and logical pretest loops, choice is a trade-off between language size, readability, and writability