

Chapter 7

Levels of Control Flow:

1. Within expressions
2. Among program units
3. Among program statements

Evolution:

- FORTRAN I control statements were based directly on IBM 704 hardware
- Much research and argument in the 1960s about the issue
- One important result: It was proven that all flowcharts can be coded with only two-way selection and pretest logical loops

Def: 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?

Chapter 7

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:

1. What is the form and type of the control expression?
2. What is the selectable segment form (single statement, statement sequence, compound statement)?
3. 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

Chapter 7

FORTRAN example:

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

ALGOL 60 if:

```
if (boolean_expr) then
begin
...
end
```

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

Chapter 7

Nested Selectors

e.g. (Pascal)

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

Which then gets the else?

Pascal's rule: else goes with the nearest then

ALGOL 60's solution - disallow direct nesting

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

Chapter 7

FORTRAN 77, Ada, Modula-2 solution - closing special words

e.g. (Ada)

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

Advantage: flexibility and readability

Modula-2 uses the same closing special word for all control structures (END)

- This results in poor readability

Chapter 7

Multiple Selection Constructs

Design Issues:

1. What is the form and type of the control expression?
2. What segments are selectable (single, compound, sequential)?
3. Is the entire construct encapsulated?
4. Is execution flow through the structure restricted to include just a single selectable segment?
5. What is done about unrepresented expression values?

Early Multiple Selectors:

1. FORTRAN arithmetic IF (a three-way selector)
IF (arithmetic expression) N1, N2, N3

Bad aspects:

- Not encapsulated (selectable segments could be anywhere)
- Segments require GOTOs

2. FORTRAN computed GOTO and assigned GOTO

Chapter 7

Modern Multiple Selectors

1. 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

Copyright © 1998 by Addison Wesley Longman, Inc. 7

Chapter 7

2. The C and 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, 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)
- default clause is for unrepresented values (if there is no default, the whole statement does nothing)

- Design choice 4 is a trade-off between reliability and flexibility (convenience)
- To avoid it, the programmer must supply a break statement for each segment

Copyright © 1998 by Addison Wesley Longman, Inc. 8

Chapter 7

3. 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 77, Modula-2, Ada)

Ada:

```

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

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

Copyright © 1998 by Addison Wesley Longman, Inc. 9

Chapter 7

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?

Counter-Controlled Loops

Design Issues:

1. What is the type and scope of the loop var?
2. What is the value of the loop var at loop termination?
3. 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?
4. Should the loop parameters be evaluated only once, or once for every iteration?

Chapter 7

1. FORTRAN 77 and 90

- *Syntax:* DO label var = start, finish [, stepsize]

- Stepsize can be any value but zero
- Parameters can be expressions

- *Design choices:*

1. Loop var can be INTEGER, REAL, or DOUBLE
2. Loop var always has its last value
3. 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
4. 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

Chapter 7

2. 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:*

1. Control expression can be int or real; its scope is whatever it is declared to be
2. Control var has its last assigned value after loop termination
3. The loop var 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

Chapter 7

3. Pascal

- Syntax:

```
for variable := init to final do  
  statement
```

- Design Choices:

1. Loop must be an ordinal type of usual scope
2. After normal termination, variable is undefined
3. The loop cannot be changed in the loop; the loop parameters can be changed, but they are evaluated just once, so it does not affect control
4. Just once

4. Ada

- Syntax:

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

Chapter 7

Ada Design choices:

1. Type of the loop var is that of the discrete range; its scope is the loop body (it is implicitly declared)
2. The loop var does not exist outside the loop
3. The loop var 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

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++) ...

Chapter 7

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

C Design Choices:

1. There is no explicit loop var
2. Irrelevant
3. Everything can be changed in the loop
4. Pretest
5. 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 function in which it is defined)

7. Java

- Differs from C++ in two ways:

1. Control expression must be Boolean
2. Scope of variables defined in the initial expression is only the loop body

Chapter 7

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)?

- 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`)
4. Ada has a pretest version, but no posttest
5. FORTRAN 77 and 90 have neither

Chapter 7

User-Located Loop Control Mechanisms

- Design issues:

1. Should the conditional be part of the exit?
2. Should the mechanism be allowed in an already controlled loop?
3. Should control be transferable out of more than one loop?

Examples:

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

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

Chapter 7

2. C, C++, and Java - `break`
Unconditional; for any loop or `switch`;
one level only (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

3. 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)) { ... }`

Chapter 7

- Perl has a built-in ~~iteration~~ arrays and hashes
e.g.,
foreach \$name (@names) { print \$name }

Unconditional Branching

Problem: readability

- Some languages do not have them: e.g., Modula-2 and Java

Label forms:

1. Unsigned int constants: Pascal (with colon)
FORTRAN (no colon)
2. Identifiers with colons: ALGOL 60, C
3. Identifiers in << ... >>: Ada
4. 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

Chapter 7

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

Chapter 7

Guarded Commands (Dijkstra, 1975)

Purpose: to support a new programming methodology (verification during program development)

1. 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)!

Chapter 7

2. 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. 320)

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
