Chapter 8

Statement-Level Control Structures
Chapter 8 Topics

• Introduction
• Selection Statements
• Iterative Statements
• Unconditional Branching
• Guarded Commands
• Conclusions
Levels of Control Flow

– Within expressions
– Among program statements
– Among program units
Control 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 algorithms represented by flowcharts can be coded with only two-way selection and pretest logical loops
Selection Statements

• A selection statement provides the means of choosing between two or more paths of execution

• Two general categories:
  – Two-way selectors
  – Multiple-way selectors
Two-Way Selection Statements

• General form:
  
  ```
  if control_expression
   then clause
   else clause
  ```

• Design Issues:
  – What is the form and type of the control expression?
  – How are the `then` and `else` clauses specified?
  – How should the meaning of nested selectors be specified?
Two-Way Selection: Examples

• FORTRAN: **IF** (boolean_expr) statement
• Problem: can select only a single statement; to select more, a **GOTO** must be used, as in the following example
  
  \begin{verbatim}
  IF (.NOT. condition) GOTO 20
  ...
  20 CONTINUE
  \end{verbatim}
• 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
Two-Way Selection: Examples

• ALGOL 60:

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

• The statements could be single or compound
Nesting Selectors

• Java example

    if (sum == 0)
      if (count == 0)
        result = 0;
      else result = 1;
    else result = 1;

• Which if gets the else?

• Java's static semantics rule: else matches with the nearest if
Nesting Selectors (continued)

• To force an alternative semantics, compound statements may be used:

```java
if (sum == 0) {
    if (count == 0)
        result = 0;
}
else result = 1;
```

• The above solution is used in java, C, C++, and C#
• Perl requires that all then and else clauses to be compound
Multiple-Way Selection Statements

• Allow the selection of one of any number of statements or statement groups

• 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: Examples

• Modern multiple selectors
  – C’s `switch` statement
    ```c
    switch (expression) {
      case const_expr_1: stmt_1;
      ...
      case const_expr_n: stmt_n;
      [default: stmt_n+1]
    }
    ```
Multiple-Way Selection: Examples

• Design choices for C’s `switch` statement
  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 (there is no implicit branch at the end of selectable segments)
     – flexible but unreliable
     – break; - restrict each execution to a single selectable segment
     – C# : every selectable segment must end with an explicit unconditional branch (e.g., break, goto)
  4. `default` clause is for unrepresented values (if there is no `default`, the whole statement does nothing)
Multiple-Way Selection: Examples

• The Ada case statement

```ada
case expression is
  when choice list => stmt_sequence;
  ...
  when choice list => stmt_sequence;
  when others => stmt_sequence;
end case;
```

• More reliable than C’s switch (once a stmt_sequence execution is completed, control is passed to the first statement following the whole case construct)
Multiple-Way Selection Using if

• Multiple Selectors can appear as direct extensions to two-way selectors, using else-if clauses, for example in Ada:

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

• It is not easily simulated with a case construct, because each selectable statement is chosen based on a Boolean expression.
Iterative Statements

• General design issues for iteration control statements:

1. How is iteration controlled?
   • counter controlled,
   • logical expression,
   • data structure

2. Where is the control mechanism in the loop?
   • Top
   • bottom
   • user-located
Counter-Controlled Loops

- A counting iterative statement has a loop variable, and a means of specifying the *initial* and *terminal*, and *stepsize* values
  - loop variable: maintain count value
  - loop parameters: Initial, terminal, stepsize

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 parameter 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: Examples

• FORTRAN 90 syntax

```fortran
DO label var = start, finish [, stepsize]
  – Step size 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 (not accessible by programmer)
     in loop, but the parameters can; since they are evaluated only once, it
     does not affect loop control
  4. Loop parameters are evaluated only once (at beginning)
Iterative Statements

• Pascal’s \texttt{for} statement

\begin{verbatim}
for variable := initial \texttt{(to|downto)} final do
statement
\end{verbatim}

• 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: Examples

- **C’s for statement**
  
  ```c
  for ([expr_1]; [expr_2]; [expr_3]) statement
  ```
  
  - Each of the expressions can be statement, or even statement sequences, with the statements separated by commas
    
    ```c
    For( count1=0, count2=1.0;
        count1 <=10 && count2 <=100.0;
        sum = ++count1 + count2, count2*= 2.5);
    ```
  
  - The second expression (i.e., the control expression):
    
    - The value of a multiple-statement expression is the value of the last statement (The zero value means false and the for is terminated).
    - The absence of control expression means true (so, infinite loop).

- There is no explicit loop variable
- Everything can be changed in the loop
- The first expression is evaluated once, but the other two are evaluated with each iteration
Iterative Statements: Logically-Controlled Loops

- Repetition control is based on a **Boolean expression**, rather than a counter
- Design issues:
  - Pre-test or post-test?
  - Should the logically controlled loop be a special case of the counting loop? or a separate statement?
- General forms:

```plaintext
while (ctrl_expr) do
  loop body
loop body
while (ctrl_expr)
```
Iterative Statements: Logically-Controlled Loops: Examples

- Pascal has separate pre-test and post-test logical loop statements (**while-do** and **repeat-until**).
- C and C++ also have both (**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**)
  - It is legal in both C and C++ to branch into both while and do-while loop bodies.
Iterative Statements: User-Located Loop Control Mechanisms

• Sometimes it is convenient for the programmers to decide a location for loop control (other than top or bottom of the loop)

• Simple design for single loops (e.g., `break`)

• For nested loops, only one loop body is exited, or the enclosing loops also be exited?
Iterative Statements: User-Located Loop Control
Mechanisms break and continue

• User-located loop control
  – Unconditional, Unlabeled break
    • C, C++, and Java: break
    • for any loop or switch; to exit one level only
  – Unconditional, labeled break
    • Java and C#: break label
    • control transfers to the label
  – An alternative: continue statement;
    • skip the rest of this iteration, but does not exit the loop
    • C, C++: Unlabeled continue
    • Java, C#: labeled continue
Iterative Statements: Iteration Based on Data Structures

- Number of elements of in a data structure control loop iteration
  - Rather than have a counter or Boolean expression
  - It uses a user-defined data structure and a user-defined function (iterator) to go through the structure’s elements

- Control mechanism
  - call to iterator function at the beginning of each iteration
  - it returns the next element in some chosen order, if there is one;
  - else loop is terminate

- C's for can be used to simulate a user-defined iterator:

  ```c
  for (p=root; p==NULL; traverse(p)){ . . . }  
  ```
Iterative Statements: Iteration Based on Data Structures (continued)

• C#’s `foreach` statement iterates on the elements of arrays and other collections:

```csharp
Strings[] strList = {"Bob", "Carol", "Ted"};
foreach (Strings name in strList)
    Console.WriteLine ("Name: {0}", name);
```

– The notation `{0}` indicates the position in the string to be displayed with the value of the first named variable (i.e., `name` in this example).
Unconditional Branching

• Transfers execution control to a specified place in the program
• Represented one of the most heated debates in 1960’s and 1970’s
• Well-known mechanism: `goto` statement
• Major concern: Readability
• Some languages do not support `goto` statement (e.g., Module-2 and Java)
• C# offers `goto` statement (can be used in `switch` statements)
• Loop exit statements are restricted and somewhat camouflaged `goto`’s
Guarded Commands

• Suggested by Dijkstra
• Purpose: to support a new programming methodology
• Basic Idea: if the order of evaluation is not important, the program should not specify one
• Basis for concurrent programming (in CSP and Ada)
Selection Guarded Command

• **Form**

  \[
  \text{if } \langle \text{Boolean exp} \rangle \rightarrow \langle \text{statement} \rangle \\
  [] \langle \text{Boolean exp} \rangle \rightarrow \langle \text{statement} \rangle \\
  \ldots \\
  [] \langle \text{Boolean exp} \rangle \rightarrow \langle \text{statement} \rangle \\
  \text{fi}
  \]

• **Semantics:** when construct is reached,
  – Evaluate all Boolean expressions
  – If more than one are true, non-deterministically choose one to execute.
  – If none are true, it is a runtime error

• **Example**

  \[
  \text{if } i = 0 \rightarrow \text{sum} := \text{sum} + i \\
  [] i > j \rightarrow \text{sum} := \text{sum} + j \\
  [] j > i \rightarrow \text{sum} := \text{sum} + i \\
  \text{fi}
  \]
Selection Guarded Command: Illustrated

- Evaluate all Boolean expressions
  - All are false
    - Run-time error
  - Exactly one is true
    - Execute associated statement
  - Randomly choose one of the true Boolean expressions

Copyright © 2006 Addison-Wesley. All rights reserved.
Loop Guarded Command

• **Form**
  
  \[ \text{do } \langle \text{Boolean} \rangle \rightarrow \langle \text{statement} \rangle \]
  
  \[ [ \ ] \langle \text{Boolean} \rangle \rightarrow \langle \text{statement} \rangle \]
  
  ...

  \[ [ \ ] \langle \text{Boolean} \rangle \rightarrow \langle \text{statement} \rangle \]

  \text{od}

• **Semantics:** for each iteration
  
  – Evaluate all Boolean expressions
  
  – If more than one are true, choose one non-deterministically; then start loop again
  
  – If none are true, exit loop
Loop Guarded Command: Illustrated

Figure 8.2
Flowgraph of the approach used with Dijkstra’s loop statement

- Evaluate all Boolean expressions
  - All are false
    - T
    - F
  - Exactly one is true
    - T
    - F
  - Randomly choose one of the true Boolean expressions
  - Execute the statement associated with the chosen Boolean expression

1-32
Conclusion

• Variety of statement-level structures
• Choice of control statements beyond selection and logical pretest loops is a trade-off between language size and writability
• Functional and logic programming languages are quite different control structures