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 units
- Among program statements
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
Control Structure

- A *control structure* is a control statement and the statements whose execution it controls.
- Design question
  - Should a control structure have multiple entries?
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

• ALGOL 60:
  ```java
  if (boolean_expr)
      then statement (then clause)
  else statement (else clause)
  ```
  The statements could be single or compound
The Control Expression

- If the then reserved word or some other syntactic marker is not used to introduce the then clause, the control expression is placed in parentheses.
- In C89, C99, Python, and C++, the control expression can be arithmetic.
- In most other languages, the control expression must be Boolean.
Clause Form

- In many contemporary languages, the then and else clauses can be single statements or compound statements.
- In Perl, all clauses must be delimited by braces (they must be compound).
- In Python and Ruby, clauses are statement sequences.
- Python uses indentation to define clauses.

```python
if x > y :
    x = y
    print "x was greater than y"
```
Nesting Selectors

- **Java example**
  
  ```java
  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:

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

- The above solution is used in C, C++, and C#
- Perl requires that all then and else clauses to be compound
• Statement sequences as clauses: Ruby

```ruby
if sum == 0 then
  if count == 0 then
    result = 0
  else
    result = 1
  end
else
  result = 1
end
end
```
Nesting Selectors (continued)

- **Python**

```python
if sum == 0 :
    if count == 0 :
        result = 0
    else :
        result = 1
```
Selector Expressions

• In ML, F#, and Lisp, the selector is an expression; in F#:

```fsharp
let y =
    if x > 0 then x
  else 2 * x
```

- If the `if` expression returns a value, there must be an else clause (the expression could produce a unit type, which has no value). The types of the values returned by then and else clauses must be the same.
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, C++, Java, and JavaScript
    ```
    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)
  4. `default` clause is for unrepresented values (if there is no `default`, the whole statement does nothing)
Multiple-Way Selection: Examples

• Ruby has two forms of case statements—we’ll cover only one

```ruby
leap = case
  when year % 400 == 0 then true
  when year % 100 == 0 then false
  else year % 4 == 0
end
```
Implementing Multiple Selectors

• Approaches:
  – Multiple conditional branches
  – Store case values in a table and use a linear search of the table
  – When there are more than ten cases, a hash table of case values can be used
  – If the number of cases is small and more than half of the whole range of case values are represented, an array whose indices are the case values and whose values are the case labels can be used
Multiple-Way Selection Using if

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

```python
if count < 10 :
    bag1 = True
elif count < 100 :
    bag2 = True
elif count < 1000 :
    bag3 = True
```
Multiple-Way Selection Using if

- The Python example can be written as a Ruby case

```ruby
begin
  when count < 10 then bag1 = true
  when count < 100 then bag2 = true
  when count < 1000 then bag3 = true
end
```
Iterative Statements

• The repeated execution of a statement or compound statement is accomplished either by iteration or recursion

• General design issues for iteration control statements:
  1. How is iteration controlled?
  2. Where is the control mechanism in the loop?
Counter–Controlled Loops

• A counting iterative statement has a loop variable, and a means of specifying the initial and terminal, and stepsizes values

• 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: Examples

• 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: Examples

• FORTRAN 95: a second form:
  
  \[ \text{name:} \text{ DO variable = initial, terminal [,stepsize]} \]
  
  \[ \text{...} \]
  
  END DO \[\text{name}\]

  – Loop variable must be an INTEGER
Iterative Statements: Examples

- **C’s for statement**
  
  ```c
  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

- 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

- It is legal to branch into the body of a for loop in C
Iterative Statements: Examples

- **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)

- **Java and C#**
  - Differs from C++ in that the control expression must be Boolean
Iterative Statements: Logically-Controlled Loops

• Repetition control is based on a Boolean
• Design issues:
  – Pre-test or post-test?
  – Should the logically controlled loop be a special case of the counting loop statement? expression rather than a counter
• 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 (\texttt{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 pre-test case (\texttt{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 \texttt{goto}
Iterative Statements: Logically-Controlled Loops: Examples

- Ada has a pretest version, but no post-test
- FORTRAN 77 and 90 have neither
- Perl has two pre-test logical loops, \texttt{while} and \texttt{until}
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`)
- Design issues for nested loops
  1. Should the conditional be part of the exit?
  2. Should control be transferable out of more than one loop?
Iterative Statements: User-Located Loop Control Mechanisms \texttt{break} and \texttt{continue}

- C, C++, and Java: \texttt{break} statement
- Unconditional; for any loop or \texttt{switch}; one level only
- Java and C# have a labeled \texttt{break} statement: control transfers to the label
- An alternative: \texttt{continue} statement; it skips the remainder of this iteration, but does not exit the loop
Iterative Statements: Iteration Based on Data Structures

• Number of elements of in a data structure control loop iteration
• Control mechanism is a call to an iterator function that returns the next element in some chosen order, if there is one; else loop is terminate
• C's for can be used to build a user-defined iterator:

```java
for (p=root; p!=NULL; traverse(p)) {
}
```

```java
Iterator e = collection.iterator();
while(e.hasNext()) {
    System.out.println(e.next());
}
```
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 (String name in strList)
    Console.WriteLine (“Name: {0}”, name);
```

• **The notation {0}** indicates the position in the string to be displayed.

```csharp
for (String name : strList )
    System.out.println(name);
```
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: \texttt{goto} statement
- Major concern: Readability
- Some languages do not support \texttt{goto} statement (e.g., Module-2 and Java)
- C# offers \texttt{goto} statement (can be used in \texttt{switch} statements)
- Loop exit statements are restricted and somewhat camouflaged \texttt{goto’s}
Guarded Commands

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

• Form

if <Boolean exp> -> <statement>
[] <Boolean exp> -> <statement>
...
[] <Boolean exp> -> <statement>
fi

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

if x >= y -> max := x
[] y >= x -> max := y
fi
Selection Guarded Command: Illustrated

Evaluate all Boolean expressions

All are false

- T: Run-time error
- F: Exactly one is true

Exactly one is true

- T: Execute associated statement
- F: Randomly choose one of the true Boolean expressions
Loop Guarded Command

• **Form**
  
  \[
  \text{do} \quad \langle \text{Boolean} \rangle \rightarrow \langle \text{statement} \rangle \\
  [ ] \quad \langle \text{Boolean} \rangle \rightarrow \langle \text{statement} \rangle \\
  \ldots \\
  [ ] \quad \langle \text{Boolean} \rangle \rightarrow \langle \text{statement} \rangle \\
  \text{od}
  \]

• **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

Exactly one is true

Randomly choose one of the true Boolean expressions

Execute the statement associated with the chosen Boolean expression
Guaraded Commands: Rationale

- Connection between control statements and program verification is intimate
- Verification is impossible with goto statements
- Verification is possible with only selection and logical pretest loops
- Verification is relatively simple with only guarded commands
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