Chapter 8 Statement-Level Control Structures

In Chapter 7, the flow of control within expressions, which is governed by operator associativity and precedence rules, was discussed. This chapter discusses flow of control among statements.

8.1 Introduction

- Within expressions (Chapter 7)
- Among program units (Chapter 9)
- Among program statements (this chapter)

8.1 Introduction (Cont'd)

- At least two additional linguistic mechanisms are necessary to make the computations in programs flexible and powerful:
  - Some means of selecting among alternative control flow paths
  - Some means of causing the repeated execution fo statements or sequences of statements

8.1 Introduction (Cont'd)

- Statements that provide these kinds of capabilities are called **control statements**
- A **control structure** is a control statement and the collection of statements whose execution it controls
- FORTRAN I control statements were based directly on IBM 704 hardware

8.1 Introduction (Cont'd)

- It was proven that all algorithms that can be expressed by flowcharts can be coded in a programming language with only two control statements
  - One for choosing between two control flow paths
    - IF-THAN-ELSE
  - One for logically controlled iterations
    - WHILE
- Bohm, Corrado; Giuseppe Jacopini (May 1966). "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules". Communications of the ACM 9 (5): 366–371.

8.2 Selection Statements

- Selection statements fall into two general categories
  - Two-way
  - N-way or multiple selection

8.2.1 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?

8.2.1.2 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

8.2.1.3 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 Fortran 95, Ada, Python, and Ruby, clauses are statement sequences.
  - The complete selection statement is terminated with a reserved word
- Python uses indentation to define clauses

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

8.2.1.4 Nesting Selectors

- Java example

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

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

- Which `if` gets the `else`?
  - It is the so-called dangling-else problem

8.2.1.4 Nesting Selectors (Cont'd)

- Solutions to dangling-else problem:
  - C, C++, C#, and Java's static semantics rule: **`else`** matches with the nearest previous **`if`**

8.2.1.4 Nesting Selectors (Cont'd)

- Solutions to dangling-else problem:
  - Perl requires that all `then` and `else` clauses be compound
    - "{" and "}" cannot be ignored

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

8.2.1.4 Nesting Selectors (Cont'd)

- Solutions to dangling-else problem:
  - Fortran 95, Ada, Ruby and Rua
    - Use of a special word to mark the end of the whole selection statement

8.2.1.4 Nesting Selectors (Cont'd)

- Statement sequences as clauses: Ruby

```
if sum == 0 then
  if count == 0 then
    result = 0
  else
    result = 1
  end
end
```

```
if sum == 0 then
  if count == 0 then
    result = 0
  end
else
  result = 1
end
```

8.2.1.5 Selector Expressions

- In ML, F#, and LISP, the selector is an expression
- F#

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

- If the **`if`** expression returns a value, there must be an else clause

8.2.1.4 Nesting Selectors (Cont'd)

- Python (By indentation)

```
if sum == 0 :
    if count == 0 :
      result = 0
    else :
      result = 1
```

```
if sum == 0 :
    if count == 0 :
      result = 0
else :
    result = 1
```

8.2.2 Multiple-Selection Statements

- The multiple-selection statement allows the selection of one of any number of statements or statement groups. It is, therefore, a generalization of a selector.
  - Two-way selectors can be built with a multiple selector.
  - Although a multiple selector can be built from two-way selectors and gotos,
    - Cumbersome, unreliable, and difficult to write and read

8.2.2 Multiple-Selection Statements (Cont'd)

- 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. How are case values specified?
  5. What is done about unrepresented expression values?

8.2.2.2 Examples of Multiple Selectors

- C, C++, Java, and JavaScript

```
switch (expression) {
  case const_expr1: stmt1;
  …
  case const_exprn: stmtn;
  [default: stmtn+1]
}
```

- The control expression and constant expressions are some discrete type

8.2.2.2 Examples of Multiple Selectors (Cont'd)

- 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)

8.2.2.2 Examples of Multiple Selectors (Cont'd)

- C#
  - Differs from C in that it has a static semantics rule that disallows the implicit execution of more than one segment
  - Each selectable segment must end with an unconditional branch (**`goto`** or **`break`**)
  - Also, in C# the control expression and the case constants can be strings
  - https://msdn.microsoft.com/zh-tw/library/aa664749(v=vs.71).aspx

8.2.2.2 Examples of Multiple Selectors (Cont'd)

- C#
  - Differs from C in that it has a static semantics rule that disallows the implicit execution of more than one segment
  - Each selectable segment must end with an unconditional branch (`goto` or `break`)
  - Also, in C# the control expression and the case constants can be strings

8.2.2.2 Examples of Multiple Selectors (Cont'd)

- Ruby:
  - The semantics is that the Boolean expressions are evaluated one at a time, top to bottom.

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

8.2.2.3 Implementing Multiple Selection Structures

- Multiple conditional branches
  - See the simple translation in P364
- 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

8.2.2.4 Multiple Selection Using `if`

- In many situations, a switch or case statement is inadequate for multiple selection
  - E.g., when selections must be made on the basis of a Boolean expression rather than some ordinal type

8.2.4 Multiple Selection Using `if`

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

```
if count < 10 :
  bag1 = True
elif count < 100 :
  bag2 = True
elif count < 1000 :
  bag3 = True
```

8.2.4 Multiple Selection Using `if`

- The Python example can be written as a Ruby **`case`**

```
case
   when count < 10 then bag1 = true
   when count < 100 then bag2 = true
   when count < 1000 then bag3 = true
end
```

8.3 Iterative Statements

- The repeated execution of a statement or compound statement is accomplished either by iteration or recursion
- An iterative statement is one that causes a statement or collections of statements to be executed zero, one, or more times
  - Loop
  - The first iterative statements in programming languages were directly related to arrays

8.3 Iterative Statements (Cont'd)

- General design issues for iteration control statements:
  1. How is iteration controlled?
  2. Where is the control mechanism in the loop?
- Some terminologies:
  - Body, pretest, posttest, iteration statement

8.3.1 Counter-Controlled Loops

- A counting iterative control statement has a variable, called the **loop variables**
  - Loop parameters
    - Initial and terminal values
    - Stepsize
- Logically controlled loops are more general than counter-controlled loops
- Counter-controlled loops are sometimes supported by machine instructions

8.3.1.1 Design Issues

1. What are the type and scope of the loop variable?
2. 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?
3. Should the loop parameters be evaluated only once, or once for every iteration?

8.3.1.2 The `for` Statement of the C-based Language

- C-based languages

  `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
- Design choices:
  - 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

8.3.1.2 The `for` Statement of the C-based Language

- 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

8.3.1.4 The `for` Statement of Python

- Python

  `for` loop_variable **`in`** object**`:`**

  - loop body

  [**`else`**`:`

  - else clause]

  – The object is often a range, which is either a list of values in brackets (`[2, 4, 6]`), or a call to the range function (**`range`**`(5)`, which returns `0, 1, 2, 3, 4`
  – The loop variable takes on the values specified in the given range, one for each iteration
  – The else clause, which is optional, is executed if the loop terminates normally

8.3.2 Logically Controlled Loops

- In many cases, collections of statements must be repeatedly executed, but the repetition control is based on a Boolean expression rather than a counter
- Design issues:
  - Pretest or posttest?
  - Should the logically controlled loop be a special case of the counting loop statement or a separate statement?

8.3.2.2 Examples

- C and C++ have both pretest and posttest forms, in which the control expression can be arithmetic:

  `while` （control_expr）**`do`**

  loop body

  `do`

  loop body

  `while` （control_expr）

  - In both C and C++ it is legal to branch into the body of a logically-controlled loop

- Java is like C and C++, except the control expression must be Boolean (and the body can only be entered at the beginning -- Java has no **`goto`**

8.3.3 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)
  - The most interesting question is whether a single loop or several nested loops can be exited
- 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?

8.3.3 User-Located Loop Control Mechanisms

- C , C++, Python, Ruby, and C# have unconditional unlabeled exits (**`break`**)
- Java and Perl have unconditional labeled exits (**`break`** in Java, **`last`** in Perl)
- C, C++, and Python have an unlabeled control statement, **`continue`**, that skips the remainder of the current iteration, but does not exit the loop
- Java and Perl have labeled versions of **`continue`**

8.3.4 Iteration Based on Data Structures

- The number of elements in a data structure controls 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 simulate a user-defined iteration statement

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

8.3.4 Iteration Based on Data Structures

- PHP
  - `current` points at one element of the array
  - `next` moves `current` to the next element
  - `reset` moves `current` to the first element

- Java 5.0 (uses **`for`**, although it is called foreach)

  For arrays and any other class that implements the `Iterable` interface, e.g., `ArrayList`

```
for (String myElement : myList) { … }
```

8.3.4 Iteration Based on Data Structures

- C# and F# also have generic library classes for collections
  - Predefined generic collections have built-in iterators that are used implicitly with the foreach statement.

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
- Major concern: Readability
- Some languages do not support **`goto`** statement (e.g., Java)
- C# offers **`goto`** statement (can be used in **`switch`** statements)