Statement Level Control Structures
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 questions
  - Should a control structure have multiple entries?
  - What control statements should a language have, beyond 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

- 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 Statements

- 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
Two-Way Selection Statements

- FORTRAN example:
  - IF (.NOT. condition) GOTO 20
  - ...
  - ...
  - 20 CONTINUE

- 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 Statements

- Examples
  - ALGOL 60 if:
  - if (boolean_expr)
  - then statement (the then clause) else statement (the else clause)
  - The statement (the else clause)
  - The statements could be single or compound
Two-Way Selection Statements

- FORTRAN 90 and Ada solution – closing special words
  - e.g. (Ada)

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

- Advantage: readability
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 languages such as Ada, Java, Ruby, and C#, 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 Fortran 95, Ada, and Ruby, clauses are statement sequences.
- Python uses indentation to define clauses.

```python
if x > y :
    x = y
print "case 1"
```
Nesting Selectors

- Java example
  
  ```java
  if (sum == 0)
    if (count == 0)
      result = 0;
    else result = 1;
  
  Which if gets the else?
  
  Java's static semantics rule: else matches with the nearest if
Nesting Selectors

- 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 C, C++, and C#
- Perl requires that all then and else clauses to be compound
Nesting Selectors

• Statement sequences as clauses:
  Ruby

  if sum == 0 then
    if count == 0 then
      result = 0
    else
      result = 1
    end
  end
end
Nesting Selectors

- Python

```python
if sum == 0 :
    if count == 0 :
        result = 0
    else :
        result = 1
else :
```
Multiple-Way Selection Statements

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

- Design Issues:
  - What is the form and type of the control expression?
  - How are the selectable segments specified?
  - Is execution flow through the structure restricted to include just a single selectable segment?
  - How are case values specified?
  - What is done about unrepresented expression values?
Multiple-Way Selection: Examples

- C, C++, and Java

```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
  - Control expression can be only an integer type
  - Selectable segments can be statement sequences, blocks, or compound statements
  - 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)
Multiple-Way Selection: Examples

- 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 (\texttt{goto} or \texttt{break})
  - Also, in C# the control expression and the case constants can be strings
Multiple-Way Selection: Examples

• 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 after the case statement)
Multiple-Way Selection: Examples

- Ada design choices:
  - Expression can be any ordinal type
  - Segments can be single or compound
  - Only one segment can be executed per execution of the construct
  - Unrepresented values are not allowed

- Constant List Forms:
  - A list of constants
  - Can include:
    - Subranges
    - Boolean OR operators (|)
Multiple-Way Selection: Examples

• Ruby has two forms of case statements
  
  1. One form uses when conditions

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

  2. The other uses a case value and when values

     ```ruby
     case in_val
       when -1 then neg_count++
       when 0 then zero_count++
       when 1 then pos_count++
       else puts "Error – in_val is out of range"
     end
     ```
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
case
  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:
  – How is iteration controlled?
  – 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 *stepsize* values.

- Design Issues:
  - What are the type and scope of the loop variable?
  - 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?
  - Should the loop parameters be evaluated only once, or once for every iteration?
Iterative Statements: Examples

- FORTRAN 95 syntax
  
  ```fortran
  DO label var = start, finish [, stepsize]
  ```

- Stepsize can be any value but zero

- Parameters can be expressions

- Design choices:
  - Loop variable must be `INTEGER`
  - 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
  - Loop parameters are evaluated only once
Iterative Statements: Examples

- FORTRAN 95: a second form:
  [name:] Do variable = initial, terminal [,stepsize]
  ...
  End Do [name]

- Cannot branch into either of Fortran’s Do statements
Iterative Statements: Examples

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

- Design choices:
  - Type of the loop variable is that of the discrete range (A discrete range is a subrange of an integer or enumeration type).
  - Loop variable does not exist outside the loop
  - The loop variable cannot be changed in the loop, but the discrete range can; it does not affect loop control.
  - The discrete range is evaluated just once

- Cannot branch into the loop body
Iterative Statements: Examples

- 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
Iterative Statements: Examples

• C++ 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 loop body)

• Java and C#
  - Differs from C++ in that the control expression must be Boolean
Iterative Statements: Examples

- 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
Iterative Statements: Logically-Controlled Loops

- Repetition control is based on a Boolean expression
- Design issues:
  - Pretest or posttest?
  - Should the logically controlled loop be a special case of the counting loop statement or a separate statement?
Iterative Statements: Logically-Controlled Loops: Examples

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

```
while (ctrl_expr) do
  loop body
end do
while (ctrl_expr)
```

• 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`
Iterative Statements: Logically-Controlled Loops: Examples

- Ada has a pretest version, but no posttest.
- FORTRAN 95 has neither.
- Perl and Ruby have two pretest logical loops, while and until. Perl also has two posttest loops.
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
  - Should the conditional be part of the exit?
  - Should control be transferable out of more than one loop?
Iterative Statements: 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**
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:
  ```c
  for (p=root; p==NULL; traverse(p))
  ```
Iterative Statements: 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**
  - For any collection that implements the `Iterator` interface
  - `next` moves the pointer into the collection
  - `hasNext` is a predicate
  - `remove` deletes an element

- **Perl**
  - built-in iterator for arrays and hashes, `foreach`
Iterative Statements: Iteration Based on Data Structures

- **Java 5.0** (uses for, although it is called foreach)
  - For arrays and any other class that implements `Iterable` interface, e.g., `ArrayList`
    
    ```java
    for (String myElement : myList) { ... }
    ```

- **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
Iterative Statements: Iteration Based on Data Structures

- Lua
  - Lua has two forms of its iterative statement, one like Fortran’s Do, and a more general form:
    ```lua
    for variable_1 [, variable_2] in iterator(table) do
      ...
    end
    ```
  - The most commonly used iterators are `pairs` and `ipairs`
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)
- Loop exit statements are restricted and somewhat camouflaged goto’s
Guarded Commands

- Designed 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
Loop Guarded Command

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

- **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
Guarded 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