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?
     • Logical expression or counter
  2. Where is the control mechanism in the loop?
     • Top or bottom
Counter–Controlled Loops

• A counting iterative statement has
  1. a loop variable, and
  2. a means of specifying the *initial* and *terminal*, and
  3. *stepsize* values
Counter–Controlled Loops

• **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?
Counting Loops: Fortran

- FORTRAN 95 syntax
  
  ```fortran
  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. 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
  3. Loop parameters are evaluated only once
Counting Loops: Fortran

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

  – Cannot branch into either of Fortran’s Do statements
Counting Loops: Ada

```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 sub-range 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**
for ([expr_1] ; [expr_2] ; [expr_3]) statement

- The expressions can be whole statements, or even statement sequences, with the statements separated by commas

- If the second expression is absent, it is an infinite loop
Counting Loops: C

- 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
Counting Loops: C–relatives

- **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
Counting Loops: Python

```python
for loop_variable in object:
    - loop body
[else:
    - else clause]
```

- The object is often a range – For example, `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
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?
• C and C++ have both pretest and posttest forms, in which the control expression can be arithmetic:

```c
while (ctrl_expr) do
    loop body
loop body
while (ctrl_expr)
```
Logically-Controlled Loops: C et al.

• Java is like C and C++, except the control expression must be **Boolean**
  - Body of the loop can only be entered at the beginning -- Java has no `goto`
Other Logically–Controlled Loops

- Ada has a pretest version, but no posttest

- FORTRAN 95 has neither

- Perl and Ruby have two pretest logical loops, `while` and `until`.

- In perl, use the `while` and `until` as expression modifiers to get post–test loop
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?
break and continue

• C, C++, Python, Ruby, and C# have unconditional unlabeled exits (break)
• Java and Perl have unconditional labeled exits (break in Java, last in Perl)
break and continue

• 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`
Iteration Based on Data Structures

- Number of elements in a data structure can 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 terminated
- C's `for` can be used to build a user-defined iterator:
  ```c
  for (p=root; p==NULL; traverse(p)) {
  }
  ```
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 has a built-in iterator for arrays and hashes, foreach
Iteration Based on Data Structures

• Java 5.0 introduced for, although it is actually foreach

• For arrays and any other class that implements an 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.
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
Unconditional Branching

- 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
Group activities

- Find two examples that showcase the *for* loop in Python – Team Merida
- Find two examples that highlight why a *do-while* loop is better than a *while* loop – Team Nemo and Remy
- Find a more efficient way to implement *switch-case* statements – Team Dory, Buzz
- Show two examples where an *iterator* would be useful – Team Wall-E and Sulley
• Write a function to add 2 numbers
  – All members of Team Dory, Nemo, Remy
• The rest of the class – You are expecting to use the function for a calculator program
  – Find a member of those teams and try to agree on a function that works for you calculator program