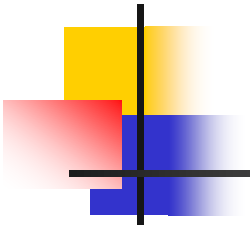
Statement-Level Control Structures

Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu



Iterative Statements

- An ***iterative statement*** is one that causes a statement or collection of statements to be executed zero, one, or more times.
- An iterative statement is often called a ***loop***.

Iterative Statements: Counter-Controlled Loops

- A counting iterative control statement has a variable, called the ***loop variable***, in which the count value is maintained.
- It also includes some means of specifying the ***initial*** and ***terminal*** values of the loop variable, and the difference between sequential loop variable values, often called the ***stepsize***.
- The initial, terminal, and stepsize specifications of a loop are called the ***loop parameters***.

Iterative Statements: The Ada for Statement

- The Ada ***for*** statement has the following form:

  for variable ***in*** [***reverse***] discrete_range ***loop***
  . . .
  end loop;

  - A discrete range is a subrange of an integer or enumeration type, such as 1..10 or Monday..Friday.
  - The ***reverse*** reserved word, when present, indicates that the values of the discrete range are assigned to the loop variable in reverse order.

Iterative Statements: The Ada for Statement

- Example:

```
for i in 1 .. 10 loop
    i := i + 1;
end loop;
```

Iterative Statements: The Ada for Statement

- The most interesting new feature of the Ada ***for*** statement is the scope of the loop variable, which is the range of the loop.
  - The variable is implicitly declared at the ***for*** statement and implicitly undeclared after loop termination.
- For example, in

```
Count : Float := 1.35;
for Count in 1..10 loop
    Sum := Sum + Count;
end loop;
```

  - the Float variable Count is unaffected by the ***for*** loop.
  - Upon loop termination, the variable Count is still Float type with the value of 1.35.
  - Also, the Float-type variable Count is hidden from the code in the body of the loop, being masked by the loop counter Count, which is implicitly declared to be the type of the discrete range, Integer.

Iterative Statements: The for Statement of the C-Based Languages

- The general form of C's for statement is

for (expression_1; expression_2; expression_3)
loop body

Iterative Statements: The for Statement of the C-Based Languages

- Following is an example of a skeletal C for statement:

```
for (count = 1; count <= 10; count++)
        . . .
}
```

  - All of the expressions of C's for are ***optional***.
  - An absent second expression is considered true, so a for without one is potentially an infinite loop.
  - If the first and/or third expressions are absent, no assumptions are made.
  - If the first expression is absent, it simply means that no initialization takes place.

Iterative Statements: The for Statement of the C-Based Languages

- C's ***for*** is more flexible than the counting loop statement of Ada, because each of the expressions can comprise multiple expressions, which in turn allow multiple loop variables that can be of any type.

- When multiple expressions are used in a single expression of a ***for*** statement, they are separated by commas.

- All C statements have values, and this form of multiple expression is no exception. The value of such a multiple expression is the value of the last component.

Iterative Statements: The for Statement of the C-Based Languages

- Consider the following for statement:

```
for (count1 = 0, count2 = 1.0;
        count1 <= 10 && count2 <= 100.0;
        sum = ++count1 + count2, count2 *= 2.5);
```

Iterative Statements: The for Statement of Python

- The general form of Python's ***for*** is

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

  - The loop variable is assigned the value in the object, which is often a range, one for each execution of the loop body.
  - The else clause, when present, is executed if the loop terminates normally.

Iterative Statements: The for Statement of Python

- Consider the following example:

```
for count in [2, 4, 6]:
    print count
```

- For most simple counting loops in Python, the range function is used. range takes one, two, or three parameters.

- The following examples demonstrate the actions of range:

```
range(5) returns [0, 1, 2, 3, 4]
range(2, 7) returns [2, 3, 4, 5, 6]
range(0, 8, 2) returns [0, 2, 4, 6]
```

Iterative Statements: 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.
- For these situations, a logically controlled loop is convenient. Actually, logically controlled loops are more general than counter-controlled loops.
- Every counting loop can be built with a logical loop, but the reverse is not true.

Iterative Statements: Logically Controlled Loops

- The C-based programming languages include both pretest and posttest logically controlled loops that are not special forms of their counter-controlled iterative statements.
- The pretest and posttest logical loops have the following forms:

```
while (control_expression)
    loop body
```

and

```
do
    loop body
while (control_expression);
```

Iterative Statements: Logically Controlled Loops

- These two statement forms are exemplified by the following C# code segments:

```
sum = 0;
indat = Int32.Parse(Console.ReadLine());
while (indat >= 0) {
    sum += indat;
    indat = Int32.Parse(Console.ReadLine());
}

value = Int32.Parse(Console.ReadLine());
do {
    value /= 10;
    digits ++;
} while (value > 0);
```