While-Loop

How are they different?

```
>> a < b < c
```

$$\underbrace{(0 < 1)}_{\underline{1}} < \underline{1} \rightarrow 0$$

$$\underbrace{(a < b)}_{\downarrow\ \text{logical } 0 \text{ or } 1} < c$$

```
>> a < b && b < c
```

Contents

1. Pop Quiz

2. Introduction to `WHILE`-Loop

3. Examples

Pop Quiz

Understanding Loops

Question 1

How many lines of output are produced by the following script?

```
for k = 100:200
    disp(k)
end
```

Same as Counting # of integers

100, 101, 102, ..., 200

A 99

B 100

[✓ C 101]

D 200

Understanding Loops

Question 2

How many lines of output are produced by the following script?

```
for k = 100:200
    if mod(k,2) == 0
        disp(k)
    end
end
```

↓

Count # of even numbers in

100, 101, 102, - - - , 200

A 50 [✓ B 51] C 100 D 101

mod(k, n) : remainder arising in division of k by n.

- $\text{mod}(7, 3) = 1 \quad \because 7 = 3 \cdot 2 + \boxed{1}$
- $\text{mod}(15, 4) = 3 \quad \because 15 = 4 \cdot 3 + \boxed{3}$

FOR-Loop: Tips

- Basic loop header:

```
for <loop var> = 1:<ending value>
```

- To adjust starting value:

```
for <loop var> = <starting value>:<ending value>
```

- To adjust step size:

```
for <loop var> = <starting value>:<step size>:<ending value>
```

Examples

- To iterate over 1, 3, 5, . . . , 9: *[step size = 2]*

```
for k = 1:2:9
```

or

1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ?

```
for k = 1:2:10
```

- To iterate over 10, 9, 8, . . . , 1: *[negative step size]*

```
for k = 10:-1:1
```

Try:

```
for k = 1 : -1 : 10
        disp(k)
end
```

Introduction to `WHILE`-Loop

Need for Another Loop

- `For`-loops are useful when the number of repetitions is known in advance.

  "Simulate the tossing of a fair coin 100 times and print the number of Heads."

- It is not very suitable in other situations such as

  "Simulate the tossing of a fair coin until the gap between the number of Heads and that of Tails reaches 10."

  We need another loop construct that terminates as soon as $|N_{\mathrm{H}} - N_{\mathrm{T}}| = 10.$

Toss :	H	H	H	T	H	T	T	H	T
N_H :	1	2	3	3	4	4	4	5	5
N_T :	0	0	0	1	1	2	3	3	4
Gap :	1	2	3	2	3	2	1	2	1

WHILE-Loop Basics

WHILE-loop is used when a code fragment needs to be executed repeatedly *while* a certain condition is true.

```
while <continuation criterion>
    <code fragment>
end
```

must evaluate to T/F.

Infinite loop

```
while true
    disp('Hello')
end
```

- The number of repetitions is *not* known in advance.
- The continuation criterion is a boolean expression, which is evaluated at the start of the loop.
  - If it is true, the loop body is executed. Then the boolean expression is evaluated again.
  - If it is false, the flow of control is passed to the end of the loop.

Simple WHILE-Loop Examples

```
k = 1; n = 10;
while k <= n
    fprintf('k = %d\n', k)
    k = k+1;
end
```

Print

k=1

k=2

⋮

k=9

k=10

Q. How would it differ if two lines in loop body are reversed?

```
k = 1;
while 2^k < 5000
    k = k+1;
end
fprintf('k = %d\n', k)
```

k=11 : $2^{11} = 2048 < 5000$

k=12 : $2^{12} = 4096 < 5000$

$\boxed{k=13}$: $2^{13} > 5000$

↑ print.

FOR-Loop to WHILE-Loop

A for-loop can be written as a while-loop. For example,

FOR

```
s = 0;
for k = 1:4
    s = s + k;
    fprintf('%2d %2d\n', k, s)
end
```

WHILE

```
k = 0; s = 0;
while k < 4
    k = k + 1; s = s + k;
    fprintf('%2d %2d\n', k, s)
end
```

$$s = 1 + 2 + 3 + 4 = \sum_{k=1}^{4} k$$

- Note that k needed to be initialized before the while-loop.
- The variable k needed to be updated inside the while-loop body.

$$s = \sum_{k=1}^{n} f(k)$$

```
s = 0;
for k = 1:n
    s = s + f(k);
end
```

Examples

Up/Down Sequence

Collatz conjecture (Ulam, Kakutani, hailstorm, (3n+1) ...)

Question

Pick a random integer between 1 and 1,000,000. Call the number n and repeat the following process:

- If n is even, replace n by $n/2$.
- If n is odd, replace n by $3n + 1$.

Does it ever take more than 1000 updates to reach 1?

- 7, 22, 11, 34, 17, ---
- 32, 16 . 8 , 4 , 2 , 1

- To generate a random integer between 1 and k, use `randi`, *e.g.*,

  (inclusive)

  ```
  randi(k)
  ```

- To test whether a number n is even or odd, use `mod`, *e.g.*,

  ```
  mod(n, 2) == 0
  ```

Another possible Syntax:

- randi([3, 10])

  rand. integer btw 3&10.

- randi([0, 1])

  Useful for coin tossing.

Attempt Using `FOR`-Loop

<main frag>

```
for step = 1:1000
    if mod(n,2) == 0
        n = n/2;
    else
        n = 3*n + 1;
    end
    fprintf(' %4d %7d\n', step, n)
end
```

- Note that once n becomes 1, the central process yields the following pattern:

$$1, 4, 2, 1, 4, 2, 1, \ldots$$

- This program continues to run even after n becomes 1.

Solution Using WHILE-Loop

```
step = 0;
while n > 1
    if mod(n,2) == 0
        n = n/2;
    else
        n = 3*n + 1;
    end
    step = step + 1;
    fprintf(' %4d %7d\n', step, n)
end
```

if n=1, n>1 evaluates to false and the loop will stop.

- This shuts down when n becomes 1!

Exercise: Gap of 10

randi([0,1])

Question

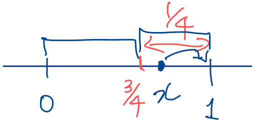
Simulate the tossing of a fair coin until the gap between the number of Heads and that of Tails reaches 10.

biased coin

	H (1)	T (0)
prob	$1/4$	$3/4$

Suggestion

- rand( )
- if-statement



```
$x \leftarrow$ rand( )

if $x < 3/4$
        $x = 0$ ;
else
        $x = 1$ ;
end
```

Summary

- For-loop is a programming construct to execute statements repeatedly.

```
for  <loop index values>
     <code fragment>
end
```

- While-loop is another construct to repeatedly execute statements. Repetition is controlled by the termination criterion.

```
while  <termination criterion is not met>
       <repeat these statements>
end
```