

# Loop Statement

NEF1104 – Problem Solving for Engineers  
Week 2: Session 6

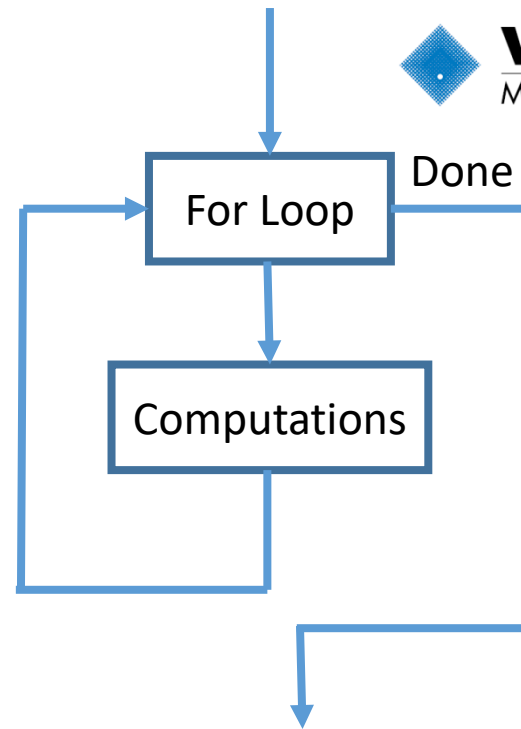
# Loop Statement

**Iteration** allows controlled repetition of a code block.

Control statements at the beginning of the code block specify the manner and extent of the repetition:

- **“for”** loop: repeat its code block a fixed number of times; largely automates the process of managing the iteration
- **“while”** loop: its code block can be repeated a variable number of times depending on the values of data being processed

# for Loop



The repeated execution of the code block is performed under the control of a loop-control variable:

- It's first set to an initial value that is tested against a terminating condition
- If the terminating test succeeds, the program leaves the “for” loop
- Otherwise, the computations in the code block are performed using the current value of that variable
- When one pass through the code block is finished, the variable is updated to its next value, and control returns to the termination test

# for loop

```
for <variable specification>  
    <code block>  
end
```

All of the mechanics of iteration control are handled automatically in the variable specification section

In some languages the variable specification is a formidable collection of statements that provide great generality of loop management

The designers of Matlab choose a much simpler approach for specifying the variable range, as shown in the general template

If we were use the variable specification  $x = 1:1:\text{length}(A)$  while  $A$  is a vector, Matlab would proceed as follows:

- Set the value of  $x$  to 1
- Evaluate the code block with  $x=1$
- Advance  $x$  to the next value 2 (because of the increment 1)
- Repeat Steps 2 and 3 until  $x$  exceeds  $\text{length}(A)$

Example: find the largest number in a row vector

```
A = floor(rand(1,10)*100); % generate a row vector of random values
theMax = A(1); % assume the 1st number is the maxima
theIndex = 1; % the index of assumed maxima is 1
for index = 1:length(A) % check all the numbers in vector A
    if A(index) > theMax % if a number is larger than "theMax"
        theMax = A(index); % this number is the new maxima
        theIndex = index; % record the index of this new maxima
    end
end
% print the result in command window, use "%d" to hold a place for numeric value
% %d: signed integer; %f: floating point; %s: character
fprintf('the largest number in A is %d at %d\n',theMax,theIndex)
```

# Breaking a **for** loop using **break**

- If you are in a for loop and find a circumstance where you really do not want to continue iterating, the **break** statement will skip immediately out the innermost containing for loop.

Example: To find the first occurrence of a number

```
A = 0:3:100;           % generate a row vector
NumberToFind = 99;      % identify the number to find
flag=0;                 % setup a variable to indicate if the number exists
for index=1:length(A)   % check all the numbers in vector A
    if A(index)==NumberToFind % if a number in A is equal to "NumberToFind"
        flag=1;           % setup the flag as "found"
        break;            % break the iteration
    end
end
if flag==1
    fprintf('the number %d is found at %d\n', NumberToFind, index);
else
    fprintf('the number %d can not be found\n', NumberToFind);
end
```

# Breaking a **for** loop using **continue**

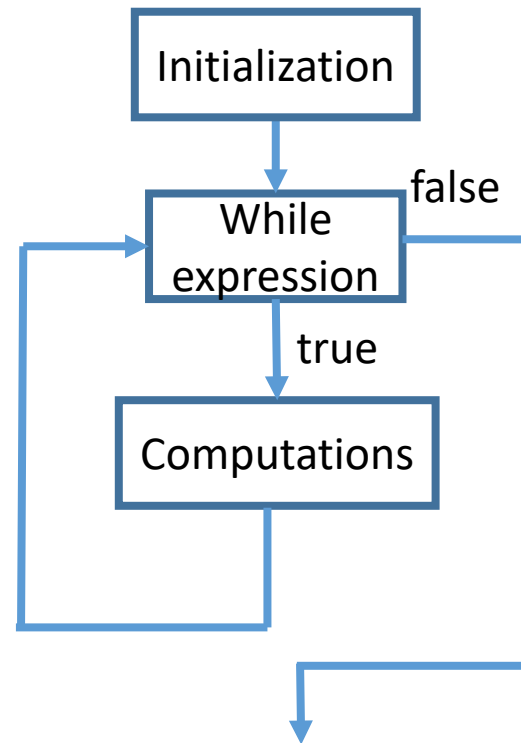
- If you want to continue iterating but omit all further steps of this iteration, you can use the **continue** statement

Example: Display the multiples of 7 from 1 through 50.

```
for i=1:50                % generate a set of numbers from 1 to 50
    if mod(i,7)~=0        % if the remainder is NOT 0 (not a multiple of 7)
        continue        % continue to the next iteration (i+1)
    else                  % if the remainder is 0 (a multiple of 7)
        fprintf('multiples of 7: %d\n', i)    % display the value
    end
end
```

Note: **mod(a,b)** returns the remainder after division a/b

# while Loop



While loops used to have more control over the number of times the iteration is repeated

- Since the termination test is performed before the loop is entered, the loop control expression must be initialized to a state that will normally permit loop entry
- It's possible that the code block is not executed at all, if there is no data to process for example




# while Loop

```
<initialization>  
while <logical expression>  
    <code block>           % must make some changes to enable the loop to terminate  
end
```

The logical expression controlling the iteration is testing some state of the workspace. Therefore, two things that were automatic in the for loop must be manually accomplished with the while loop:

- initializing the test
- updating the workspace so that the test will eventually fail and the iteration will stop

```
for index = a:b:c  
    ...  
end
```



```
index = a;  
while index<=c  
    ...  
    index = index + b;  
end
```

The loop starts at **index = a**,  
Every second iteration has an **index increased by b**  
The last iteration has **index = c**

# while Loop



<initialization>

**while** <logical expression>

<code block>

*% must make some changes to enable the loop to terminate*

**end**

Example: find the largest number in a row vector

```
A = floor(rand(1,10)*100); % generate a row vector
theMax=A(1); % assume the 1st number is the maxima
theIndex=1; % the index of assumed maxima is 1
index=1; % define the variable "index" for the position of each value
while index<=length(A) % the loop finishes when index exceeds the maximum length
    if A(index)>theMax % if a number is larger than "theMax"
        theMax=A(index); % this number is the new maxima
        theIndex=index; % record the index of this new maxima
    end
    index=index+1; % increase the index to move to the next number
end
fprintf('the largest number in A is %d at %d\n',theMax,theIndex)
```

# Breaking a **while** loop

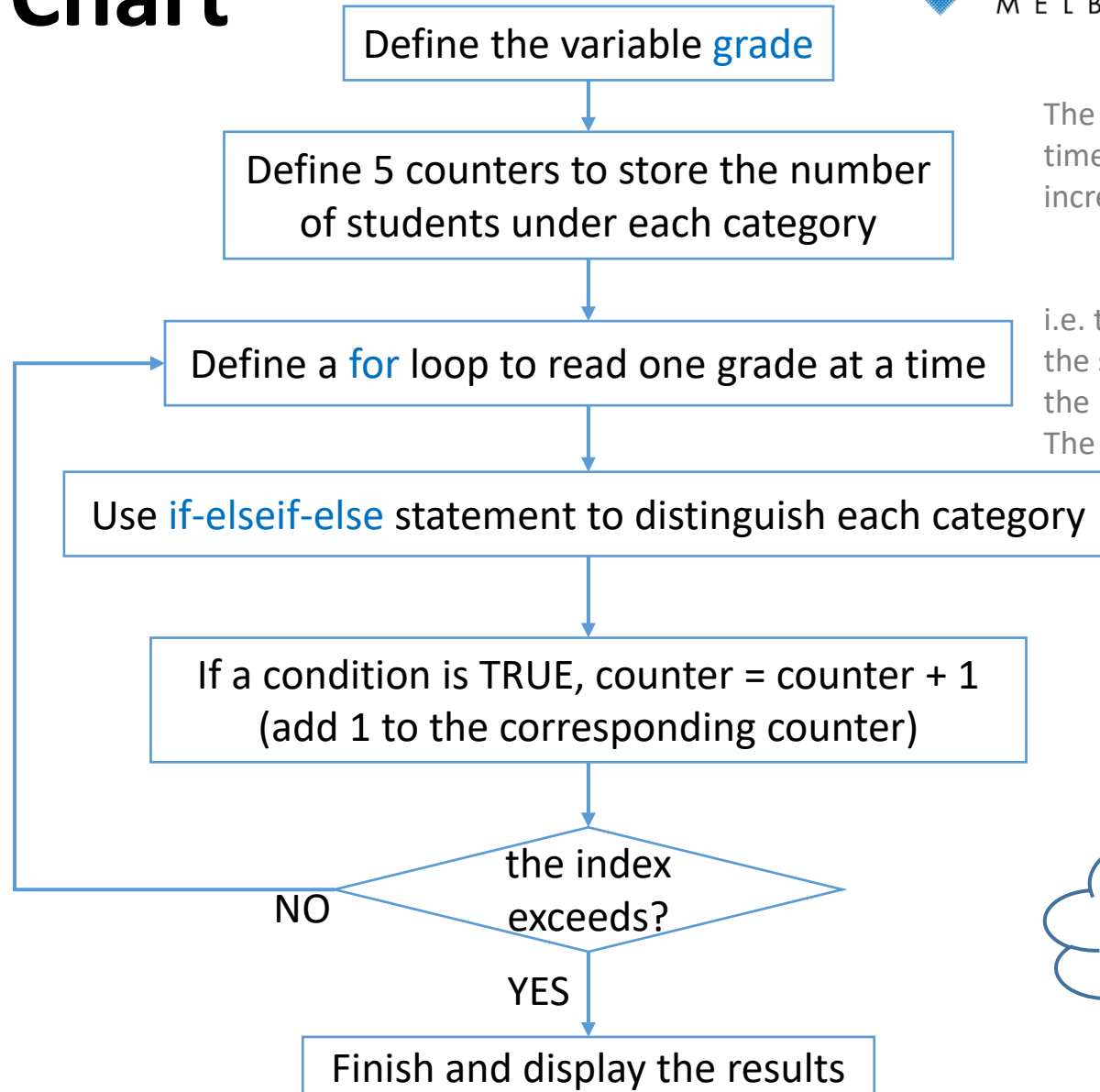
As with for loop, **break** will exit the innermost while loop,  
and **continue** will skip to the end of the loop but remain with it

# Exercise

Assume students' semester results are save in a variable `grade = floor(rand(1,50)*100)` , count the number of students who receive “HD” , “D”, ”C”, “P”, and “F” respectively.

HD	$\text{grade} \geq 80$
D	$70 \leq \text{grade} < 80$
C	$60 \leq \text{grade} < 70$
P	$50 \leq \text{grade} < 70$
F	$\text{grade} < 50$

# Flow Chart



The original counter = 0. Every time when a condition is TRUE, increase the counter by 1.

i.e. the first student's grade is `grade(1)`, the second student's grade is `grade(2)` ... the last student's result is `grade(100)`. The *i*-th student's grade is `grade(i)`.





```
grade = floor(rand(1,50)*100); % define a 1x50 vector to store 50 students' grades
counter = zeros(1,5); % initialize a 1x5 vector of zeros to count the number of HD, number
of D, ... respectively
for i = 1:length(grade) % define the iteration to take grade(1), grade(2), ... grade(100)
    if grade(i) >= 80 % HD
        counter(1) = counter(1) + 1; % HD counter +1
    elseif grade(i) >= 70 & grade(i) < 80 % D
        counter(2) = counter(2) + 1; % D counter +1
    elseif grade(i) >= 60 & grade(i) < 70 % C
        counter(3) = counter(3) + 1; % C counter +1
    elseif grade(i) >= 50 & grade(i) < 60 % P
        counter(4) = counter(4) + 1; % P counter +1
    else % all other cases (F)
        counter(5) = counter(5) + 1; % F counter +1
    end % close the if-else statement
end % close the for loop
fprintf('num of HD: %d, num of D: %d, num of C: %d, num of P: %d, num of F: %d',
counter(1), counter(2), counter(3), counter(4), counter(5));
% display results in Command Window
```