Lecture 5: Programming using MATLAB

Dr. Mohammed Hawa
Electrical Engineering Department
University of Jordan

EE201: Computer Applications. See Textbook Chapter 4.

Algorithms and Control Structures

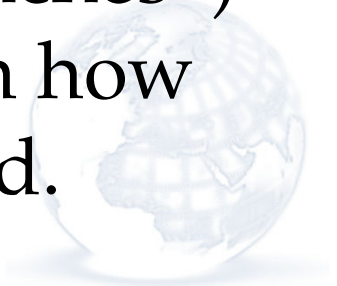
- **Algorithm**: a sequence of instructions that performs some task in a finite amount of time.
- The algorithm uses a *control structure* to execute instructions in a certain order.
- Control structure categories:
  - **Sequential operations**: Instructions executed in order.
  - **Conditional operations**: First ask a question to be answered with a true/false answer and then select the next instruction based on the answer.
  - **Iterative operations (loops)**: Repeat the execution of a block of instructions.

Before Programming

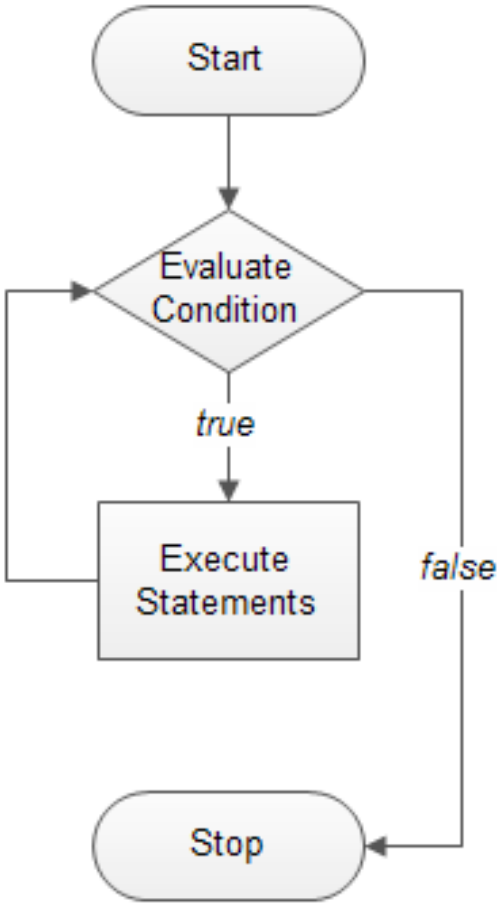
- Before writing a program, we need a plan.
- A plan helps us focus on the problem, not the code.
- Common methods to show a plan are:
  - **Flowchart**: A graphical description of the program flow.
  - **Pseudocode**: A verbal description of the program details.

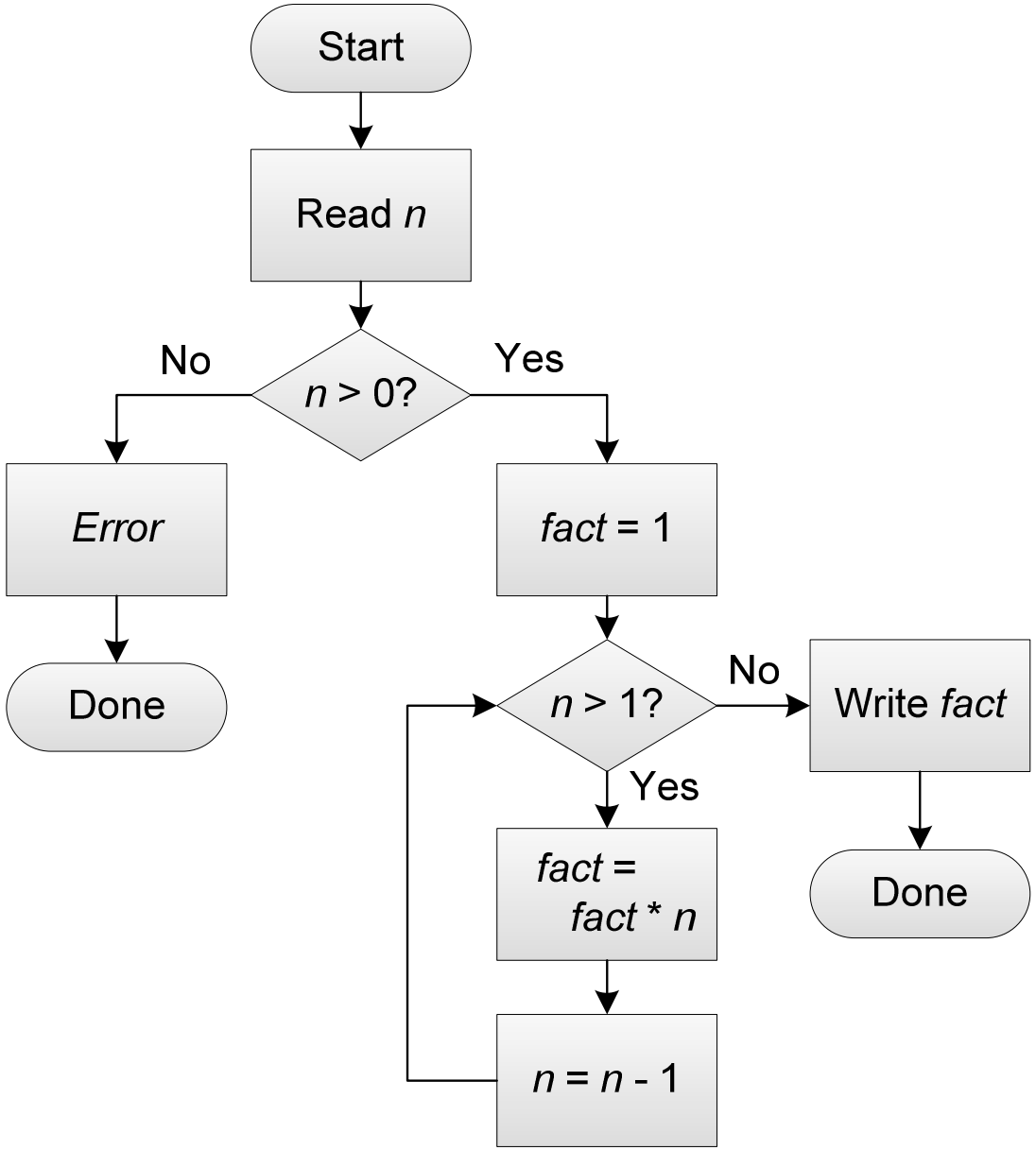
Flowcharts

- Flowcharts are geometric symbols to describe the program steps.
- They capture the "flow" of the program.
- Flowcharts are useful for developing and documenting programs that contain conditional statements, because they can display the various paths (called "branches") that a program can take, depending on how the conditional statements are executed.

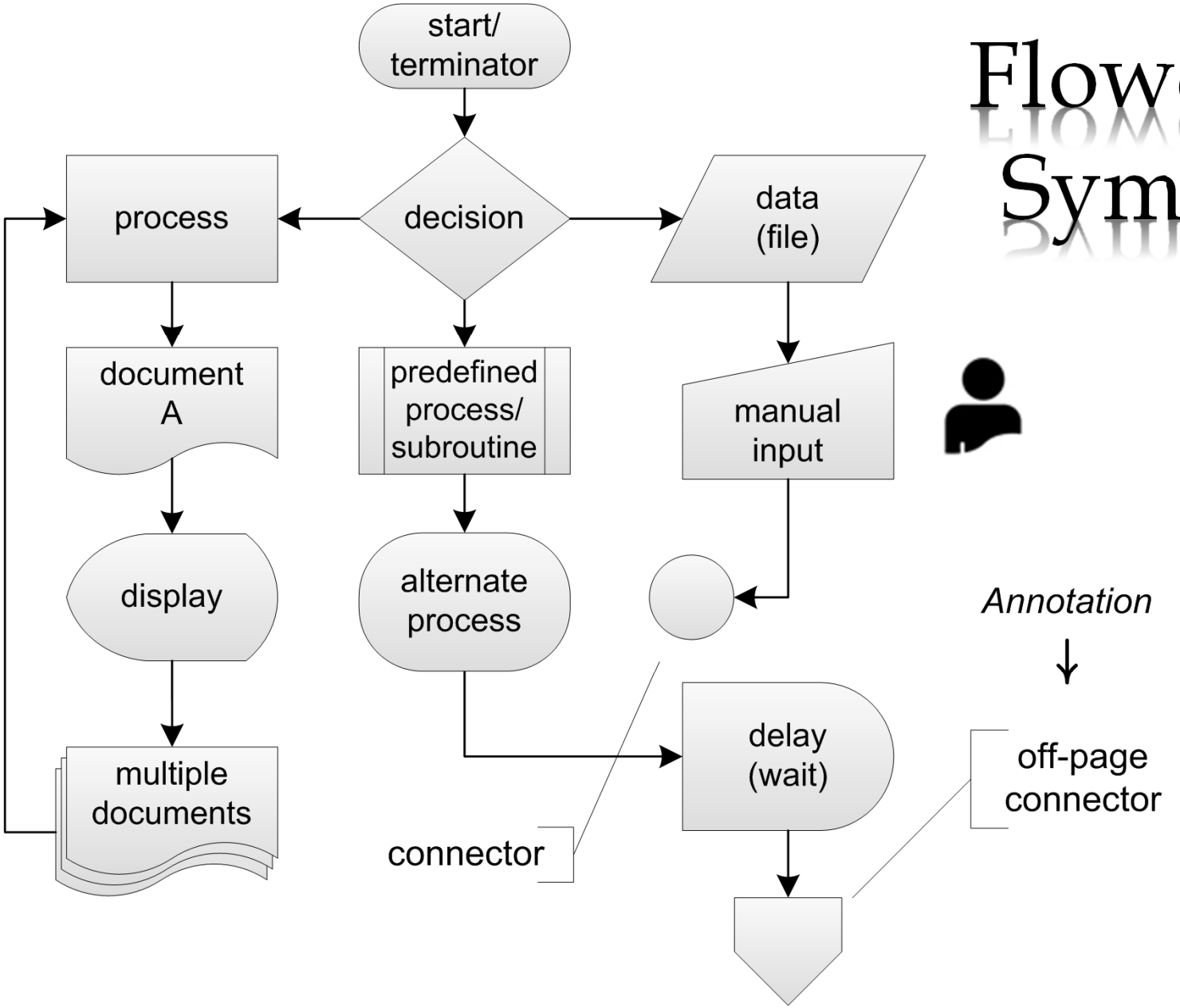


Examples





Flowchart Symbols



Pseudocode

- In pseudocode, natural language and mathematical expressions are used to construct statements that look like computer statements but without detailed syntax.
- Each pseudocode instruction may be numbered, but should be unambiguous and computable.
- Similar to a recipe.

Pseudocode Example

Input: A nonempty string of characters $S_1 S_2 \ldots S_n$, and a positive integer n giving the number of characters in the string.
Output: See the related problem below.
Procedure:

```
1   Get n
2   Get S_1 S_2 ... S_n
3   Set count = 1
4   Set ch = S_1
5   Set i = 2
6   While i ≤ n
7      If S_i equals ch
8         Set count = count + 1
9      Set i = i + 1
10  Print ch, ' appeared ', count, ' times.'
11  Stop
```

Problem 1.1 *What is printed if the input string is* `pepper`*?*

Problem 1.2 *What is printed if the input string is* `CACCTGGTCCAAC`*?*

Algorithm Distribute

Input: $(G^*, f, edge)$, where $G^* = (N, M, s, t, E^*, w)$, f is a set of flows f_e^v and $edge$ is the edge that is being distributed.

0. Initialize $scan(v) = 0, label(v) = 0, scan(e) = 0, label(e) = 0$ for all $v \in N, e \in M$
1. $vert = 0,\ capvert = 0$
2. $label(edge) = 1,\ pathcap(edge) = w(edge)$
3. **while** $(w(edge) > \sum_v f_{edge}^v)$ **or** not all labeled nodes have been scanned
4. **for** all labeled $e \in M$, with $scan(e) = 0$
5. label unlabeled neighbors of e (i.e $v \in N$)
6. $scan(e) = 1,\ pred(v) = e,\ pathcap(v) = pathcap(e)$
7. **endfor**
8. **for all** labeled $v \in N$ with $scan(v) = 0$
9. **if** $\min(w(v) - \sum_e f_e^v,\ pathcap(v)) > capvert$ **then**
10. $vert = v,\ capvert = min(w - \sum_e f_e^v,\ pathcap(v))$
11. **else**
12. label all unlabeled $e' \in M$ s.t $f_{e'}^v > 0$
13. **endif**
14. $scan(v) = 1$
15. **endfor**
16. **if** $vert > 0$ **then**
17. An augmenting path from s to t has been found: backtrack from $vert$ using $pred()$ and change the values of f_e^v as requirted.
18. **for all** $e \in M$, $v \in N$
19.. $label(e) = 0, scan(e) = 0, label(v) = 0, scan(v) = 0$
20. **endfor**
21. $vert = 0, capvert = 0, label(edge) = 1$
22. $pathcap(edge) = w(edge) - \sum_v f_{edge}^v$
23. **endif**
24. **endwhile**

During and After Programming

- Make sure to provide effective documentation along with the program. This can be accomplished using:
  - Proper selection of variable names to reflect the quantities they represent.
  - Using comments within the program.
- Debugging a program is the process of finding and removing the "bugs" or errors in a program.

Bugs

Bugs usually fall into one of two categories:

1. **Syntax errors:** such as omitting a parenthesis or comma, or spelling a command name incorrectly. MATLAB usually detects the more obvious errors and displays a message describing the error and its location.
2. Errors due to an incorrect mathematical procedure. These are called **runtime errors**. They do not necessarily occur every time the program is executed; their occurrence often depends on the particular input data. A common example is division by zero.

Finding Bugs: Debugging

To locate runtime errors, try the following:

1. Always test your program with a simple version of the problem, whose answers can be checked by hand calculations.
2. Display any intermediate calculations by removing semicolons at the end of statements.

Relational Operators

Operator	Meaning
<	Less than.
<=	Less than or equal to.
>	Greater than.
>=	Greater than or equal to.
==	Equal to.
~=	Not equal to.

Examples

```
>> a = 3;
>> b = 4;
>> a == b
ans =
     0

>> a ~= b
ans =
     1

>> a < b
ans =
     1

>> b >= -4
ans =
     1
```

```
>> x = [6 3 9];
>> y = [14 2 9];
>> z = (x < y)
z =
     1     0     0

>> z = x ~= y
z =
     1     1     0

>> z = x > 8
z =
     0     0     1
```

Relational operators can be used for array addressing.

For example

```
>> x = [6,3,9];
>> y = [14,2,9];
>> x<y
ans =
     1     0     0
>> z = x(x<y)
z =
     6
```

finds all the elements in x that are less than the corresponding elements in y. The result is `z = 6`.

The arithmetic operators +, -, *, /, and \ have precedence over the relational operators. Thus the statement

```
z = 5 > 2 + 7
```

is equivalent to

```
z = 5 > (2+7)
```

and returns the result `z = 0`.

We can use parentheses to change the order of precedence; for example, `z = (5 > 2) + 7` evaluates to `z = 8`.

The logical Class

When the relational operators are used, such as

```
x = (5 > 2)
```

they create a *logical* variable, in this case, `x`.

Logical variables may have only the values 1 (true) and 0 (false).

Just because an array contains only 0s and 1s, however, it is not necessarily a logical array. For example, in the following session `k` and `w` appear the same, but `k` is a logical array and `w` is a numeric array, and thus an error message is issued.

```
>>x = -2:2;
>>k = (abs(x)>1)
k =
    1    0    0    0    1
>>z = x(k)
z =
   -2    2
>>w = [1,0,0,0,1]; v = x(w)
??? Subscript indices must either be real
  positive... integers or logicals.
```

Accessing Arrays Using Logical Arrays

When a logical array is used to address another array, it extracts from that array the elements in the locations where the logical array has 1s.

So typing `A(B)`, where `B` is a logical array of the same size as `A`, returns the values of `A` at the indices where `B` is 1.

Given `A =[5,6,7;8,9,10;11,12,13]` and `B = logical(eye(3))`, we can extract the diagonal elements of `A` by typing `C = A(B)` to obtain `C = [5;9;13]`.

See our earlier discussion of logical indexing.

Logical Operators

Operator	Name	Definition
`~`	NOT	`~A` returns an array the same dimension as `A`; the new array has ones where `A` is zero and zeros where `A` is nonzero.
`&`	AND	`A & B` returns an array the same dimension as `A` and `B`; the new array has ones where both `A` and `B` have nonzero elements and zeros where either `A` or `B` is zero.
`|`	OR	`A | B` returns an array the same dimension as `A` and `B`; the new array has ones where at least one element in `A` or `B` is nonzero and zeros where `A` and `B` are both zero.
`&&`	Short-Circuit AND	Short-circuiting means the second operand (right hand side) is evaluated only when the result is not fully determined by the first operand (left hand side) `A & B` (A and B are evaluated) `A && B` (B is only evaluated if A is true)
`||`	Short-Circuit OR	`|` can operate on arrays but `||` only operates on scalars

Examples

```
>> a = 3;
>> b = 4;
>> c = 5;
>> x = ~(a == b)
x =
     1

>> (a < b) & (b < c)
ans =
     1

>> (a < b) && (b < c)
ans =
     1

>> 5 && 0
ans =
     0

>> [1 2] && [3 4]
??? Operands to the || and && operators must
be convertible to logical scalar values.
```

Order of precedence for operators

Precedence	Operator type
First	Parentheses; evaluated starting with the innermost pair.
Second	Arithmetic operators and logical NOT (~); evaluated from left to right.
Third	Relational operators; evaluated from left to right.
Fourth	Logical AND.
Fifth	Logical OR.

Logical functions

Logical function	Definition
`ischar(A)`	Returns a 1 if `A` is a character array and 0 otherwise.
`isempty(A)`	Returns a 1 if `A` is an empty matrix and 0 otherwise.
`isinf(A)`	Returns an array of the same dimension as `A`, with ones where `A` has '`inf`' and zeros elsewhere.
`isnan(A)`	Returns an array of the same dimension as `A` with ones where `A` has '`NaN`' and zeros elsewhere. ('`NaN`' stands for "not a number," which means an undefined result.)

Logical Functions

`isnumeric(A)`	Returns a 1 if A is a numeric array and 0 otherwise.
`isreal(A)`	Returns a 1 if A has no elements with imaginary parts and 0 otherwise.
`logical(A)`	Converts the elements of the array A into logical values.
`xor(A,B)`	Returns an array the same dimension as A and B; the new array has ones where either A or B is nonzero, but not both, and zeros where A and B are either both nonzero or both zero.

Logical Operators and the `find` Function

Consider the session

```
>> x = [5, -3, 0, 0, 8];
>> y = [2, 4, 0, 5, 7];
>> x&y
ans =
     1     1     0     0     1
>> z = find(x&y)
z =
   1    2    5
```

Note that the find function returns the *indices*, and not the *values*.

Conditional Statements: The `if` Statement

The `if` statement's basic form is

```
if logical expression
   statements
end
```

Every `if` statement must have an accompanying `end` statement. The end statement marks the end of the *statements* that are to be executed if the *logical expression* is true.

The `else` Statement

The basic structure for the use of the `else` statement is

```
if logical expression
  statement group 1
else
  statement group 2
end
```

When the test, if *logical expression,* is performed, where the logical expression may be an *array,*
the test returns a value of true only if *all* the elements of the logical expression are true!

The `elseif` Statement

The general form of the `if` statement is

```
if logical expression 1
     statement group 1
elseif logical expression 2
     statement group 2
else
     statement group 3
end
```

The `else` and `elseif` statements may be omitted if not required. However, if both are used, the `else` statement must come after the `elseif` statement to take care of all conditions that might be unaccounted for.

Exercise

File: test.m

```
a = 5;
b = 4;

if a == b
    disp(a);
    disp(b);
elseif a < b
    disp(a);
else
    disp(b);
end
```

Matlab command prompt

```
>> test
     4
>>
```

Example

- Suppose that we want to compute y, which is given by the equation:

$$y = \begin{cases} 15\sqrt{4x} + 10 & \text{if } x \geq 9 \\ 10x + 10 & \text{if } 0 \leq x < 9 \\ 10 & \text{if } x < 0 \end{cases}$$

```
function y = test(x)
if x >= 9
  y = 15*sqrt(4*x) + 10
elseif x >= 0 % already less than 9
  y = 10*x + 10
else
  y = 10
end
```

Example: if we fail to recognize how the test works, the following statements do not perform the way we might expect.

```
x = [4 -9 25];
if x < 0
   disp('Cant find square root of negative.')
else
   y = sqrt(x)
end
```

When this program is run it gives the result

```
y =
  2   0 + 3.000i   5
```

Instead, consider what happens if we test for `x` positive.

```
x = [4, -9, 25];
if x >= 0
  y = sqrt(x)
else
  disp('Cant find square root of negative.')
end
```

When executed, it produces the following message:

```
Cant find square root of negative.
```

The test if `x < 0` is false, and the test if `x >= 0` also returns a false value because `x >= 0` returns the vector `[1,0,1]`.

Loops

- Often in your programs you will want to "loop"
  - repeat some commands multiple times
- If you know how many times you want to loop
  - use a `for` loop
- If you want to loop until something happens (a condition is satisfied)
  - use a `while` loop
- If you find yourself typing similar lines more than a couple of times, use a loop

`for` Loops

A simple example of a for loop is:

```
m = 0;
x(1) = 10;
for k = 2:3:11;
  m = m + 1;
  x(m+1) = x(m) + k^2;
end
```

k takes on the values 2, 5, 8, 11. The variable m indicates the index of the array x. When the loop is finished the array x will have the values x(1)=14, x(2)=39, x(3)=103, x(4)=224.

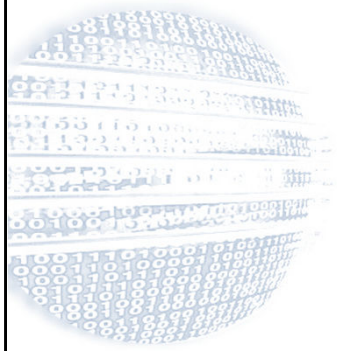
Note the following rules when using for loops with the loop variable expression `k = m:s:n`:

- The step value `s` may be negative.
  Example: `k = 10:-2:4` produces `k` = 10, 8, 6, 4.
- If `s` is omitted, the step value defaults to 1.
- If `s` is positive, the loop will not be executed if `m` is greater than `n`.
- If `s` is negative, the loop will not be executed if `m` is less than `n`.
- If `m` equals `n`, the loop will be executed only once.
- If the step value s is not an integer, round-off errors can cause the loop to execute a different number of passes than intended.

Exercise

File: loop.m

```
for i = 1:1:5

    disp(i)

end
```



Matlab command prompt

```
>> loop
     1
     2
     3
     4
     5
>>
```

Strings and Conditional Statements

A *string* is a variable that contains characters. Strings are useful for creating input prompts and messages and for storing and operating on data such as names and addresses.

To create a string variable, enclose the characters in single quotes. For example, the string variable name is created as follows:

```
>>name = 'Mohammed Ali'
name =
    Mohammed Ali
```

The following string, `number`, is *not* the same as the variable number created by typing number = 123.

```
>>number = '123'
number =
    123
```

The following prompt program is a script file that allows the user to answer *Yes* by typing either `Y` or `y` or by pressing the **Enter** key. Any other response is treated as a No answer.

```
response = input('Continue? Y/N [Y]: ','s');
if (isempty(response))|(response ==
'Y')|(response == 'y')
  response = 'Y'
else
  response = 'N'
end
```

Programming Exercise #1

- Write a MATLAB program that does the following:
- The program asks you to enter your name.
- It waits for you to enter your name and hit Enter.
- The program reads your name, counts its characters and any blank spaces in the name, then displays something like this:
- You name is “Mohammed Ali”. It has 11 characters and 1 blank space.

Using loops is slower than arrays in MATLAB

We can use the mask technique to compute the square root of only those elements of `A` that are no less than 0 and add 50 to those elements that are negative. The program is

```
A = [0, -1, 4; 9, -14, 25; -34, 49, 64];
C = (A >= 0);
A(C) = sqrt(A(C))
A(~C) = A(~C) + 50
```

`while` Loops

The `while` loop is used when the looping process terminates because a specified condition is satisfied, and thus the number of passes is not known in advance.
A simple example of a while loop is

```
x = 5;
while x < 25
   disp(x)
   x = 2*x - 1;
end
```

The results displayed by the `disp` statement are 5, 9, 17.

The typical structure of a while loop follows.

```
while logical expression
    statements
end
```

For the `while` loop to function properly, the following two conditions must occur:

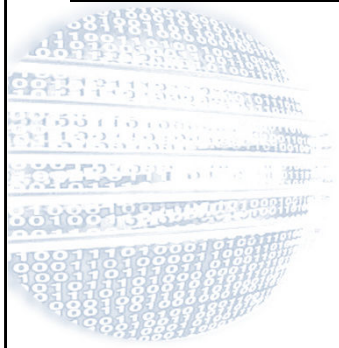
1. The loop variable must have a value before the while statement is executed.

2. The loop variable must be changed somehow by the *statements.*

Exercise

File: loop2.m

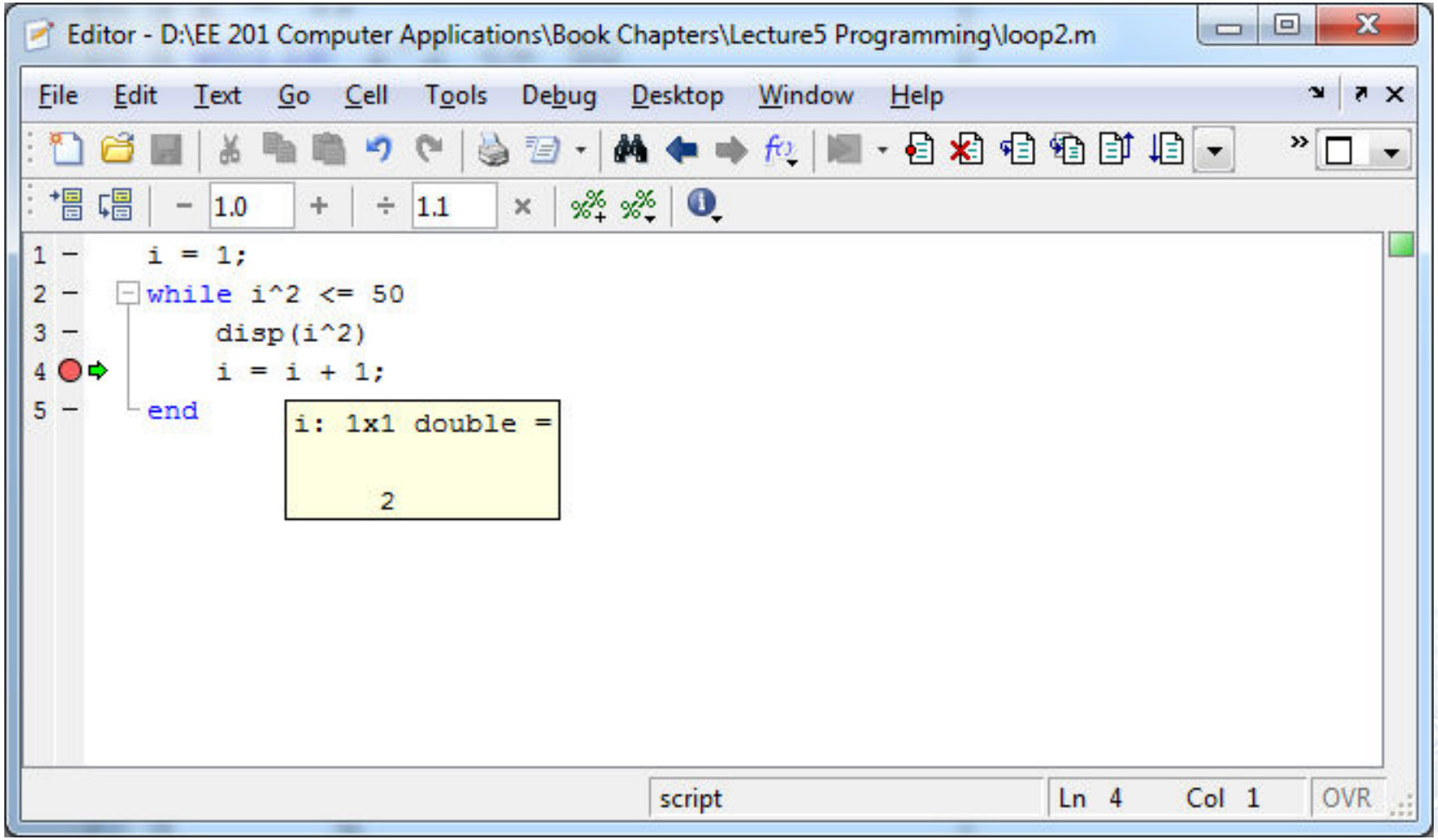
```
i = 1;
while i^2 <= 50

    disp(i^2)
    i = i + 1;

end
```



Matlab command prompt

```
>> loop2
     1
     4
     9
    16
    25
    36
    49
>>
```

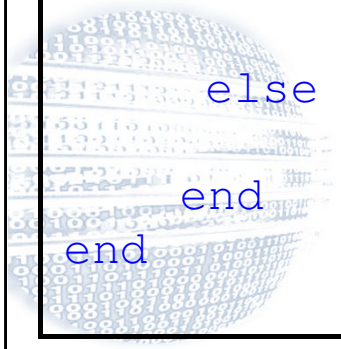
Editor/Debugger containing program to be analyzed



The `break` statement

- `break` terminates the execution of a loop, so if you have a nested loop, break will only quit the innermost loop, and the program will continue running.

```
s=6;                % initialize s to 6
while s~=1          % as long as s is not equal to 1 stay in loop
     if s==17       % if s equals 17
          sprintf('Found 17 in the loop!!')
          break;
     end
     if mod(s,2) % the actual "brains" of the iteration
          s=s/2;
     else
          s=3*s+1;
     end
end
```



The `continue` statement

The following code uses a `continue` statement to avoid computing the logarithm of a negative number.

```
x = [10,1000,-10,100];
y = NaN*x;
for k = 1:length(x)
  if x(k) < 0
    continue
  end
  y(k) = log10(x(k));
end
y
```

The result is `y = [1  3  NaN  2]`.

Homework

- Write a script file to determine how many terms are required for the sum of the series $5k^2 - 2k, k = 1, 2, 3, \ldots$ to exceed 10,000. What is the sum for this many terms?

```
total = 0; k = 0;
while total < 1e4
      k = k + 1;
      total = total + 5*k^2 - 2*k;
end
disp('The number of terms is:')
disp(k)
disp('The sum is:')
disp(total)
```

- The sum is 10,203 after 18 terms.

Exercise: Fourier Series

- $x(t) = c_0 + \sum_{n=1}^{\infty} c_n \cos(n\omega_0 t - \theta_n)$
- Discover the following periodic function:
- $x(t) = 0.5 + \frac{2}{\pi}\left[\cos(t) + \frac{1}{3}\cos(3t) + \frac{1}{5}\cos(5t) + \frac{1}{7}\cos(7t) + \cdots\right]$
- Use a `for` or `while` loop. Use `n` as the loop parameter to add certain terms then plot the result versus time $-10 \leq t \leq 10$.
- On one figure, draw the result of 3 terms.
- On one figure, draw the result of 10 terms.
- On one figure, draw the result of 100 terms.

Infinite Loops

- "Infinite loop" = piece of code that will execute again and again … without ever ending.
- Possible reasons for infinite loops:
  - getting the conditional statement wrong
  - forgetting the update step
- If you are in an infinite loop then ctrl-c stops MATLAB executing your program.

The `switch` statement

The `switch` statement provides an alternative to using the `if`, `elseif`, and `else` commands.

Anything programmed using `switch` can also be programmed using `if` statements.

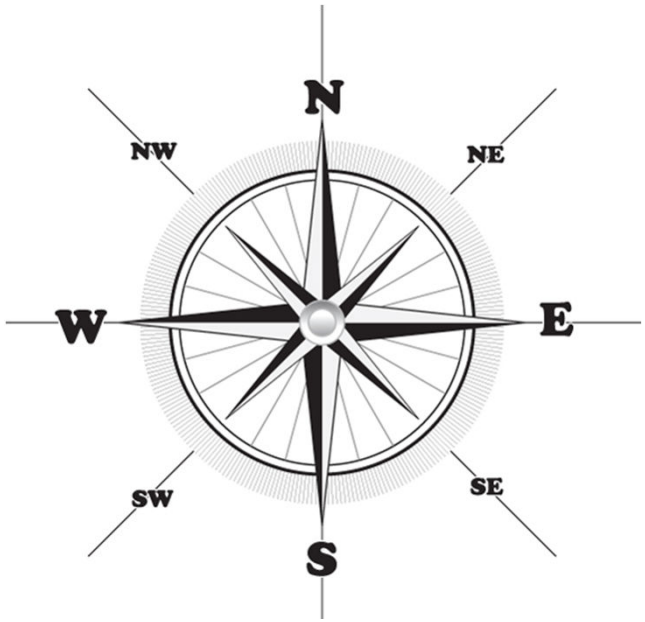
However, for some applications the `switch` statement is more readable than code using the `if` structure.

Syntax of `switch`

```
switch input expression (can be a scalar or string).
  case value1
      statement group 1
  case value2
      statement group 2
  .
  .
  .
  otherwise
      statement group n
end
```

The following switch block displays the point on the compass that corresponds to that angle.

```
switch angle
  case 45
    disp('Northeast')
  case 135
    disp('Southeast')
  case 225
    disp('Southwest')
  case 315
    disp('Northwest')
  otherwise
    disp('Direction Unknown')
end
```



Boolean Variables

- MATLAB allows boolean variables that take true/false values

```
if (atUniversity & stillAStudent)
    needMoreMoney = true;
end
```

Programming Exercise #2

- Write a MATLAB program to solve this:
- One investment opportunity pays 5.5% annual profit, while a second investment opportunity pays 4.5% annual profit.
- Determine how much longer it will take to accumulate at least $50,000 in the second investment opportunity compared to the first if you invest $1000 initially and $1000 at the end of each year.

Programming Exercise #3

- Write a MATLAB program that asks you for a hexadecimal integer number.
- The program should read that number and convert it to decimal.
- Example: 84CD hexadecimal is 33997 decimal.
- Can you improve on your program so it accepts binary or hexadecimal or decimal and converts it to all other formats? You need to accept numbers written in something like this: 94CA**h** or 110110001**b**.

Homework

- Solve as many problems from Chapter 4 as you can
- Suggested problems:
- 4.2, 4.4, 4.5, 4.11, 4.13, 4.15, 4.16, 4.17, 4.23, 4.24, 4.25, 4.26, 4.33, 4.37, 4.39, 4.47