



MATLAB

for Engineering Applications
Fifth Edition

William J. Palm III

© McGraw Hill LLC. All rights reserved. No reproduction or distribution without the prior written consent of McGraw Hill LLC.

Chapter 04

Programming with MATLAB

© McGraw Hill LLC

2

Algorithms and Control Structures

Algorithm: an ordered sequence of precisely defined instructions that performs some task in a finite amount of time. *Ordered* means that the instructions can be numbered, but an algorithm must have the ability to alter the order of its instructions using a *control structure*. There are three categories of algorithmic operations:

Sequential operations: Instructions executed in order.

Conditional operations: Control structures that 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): Control structures that repeat the execution of a block of instructions.

Structured Programming

A technique for designing programs in which a hierarchy of *modules* is used, each having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure.

In MATLAB these modules can be built-in or user-defined functions.

Advantages of Structured Programming ₁

1. Structured programs are easier to write because the programmer can study the overall problem first and then deal with the details later.
2. Modules (functions) written for one application can be used for other applications (this is called *reusable* code).
3. Structured programs are easier to debug because each module is designed to perform just one task and thus it can be tested separately from the other modules.

Advantages of Structured Programming ₂

4. Structured programming is effective in a teamwork environment because several people can work on a common program, each person developing one or more modules.
5. Structured programs are easier to understand and modify, especially if meaningful names are chosen for the modules and if the documentation clearly identifies the module's task.

Steps for Developing a Computer Solution ₁

1. State the problem concisely.
2. Specify the data to be used by the program. This is the “input.”
3. Specify the information to be generated by the program. This is the “output.”
4. Work through the solution steps by hand or with a calculator; use a simpler set of data if necessary.

Steps for Developing a Computer Solution ₂

5. Write and run the program.
6. Check the output of the program with your hand solution.
7. Run the program with your input data and perform a reality check on the output.
8. If you will use the program as a general tool in the future, test it by running it for a range of reasonable data values; perform a reality check on the results.

Effective documentation can be accomplished with the use of

1. Proper selection of variable names to reflect the quantities they represent.
2. Use of comments within the program.
3. Use of structure charts.
4. Use of flowcharts.
5. A verbal description of the program, often in *pseudocode*.

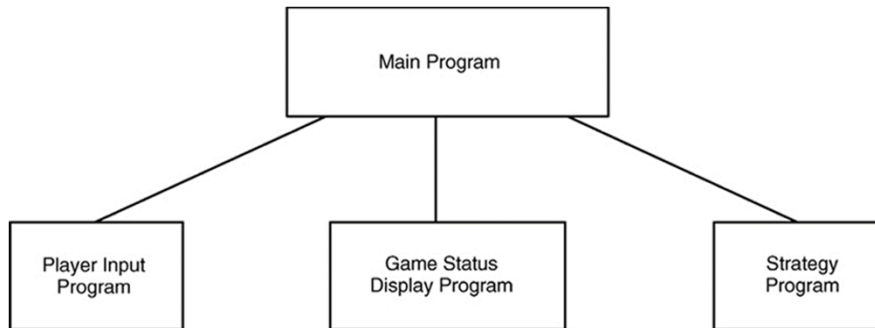
Documenting with Charts

Two types of charts aid in developing structured programs and in documenting them.

These are *structure charts* and *flowcharts*.

A structure chart is a graphical description showing how the different parts of the program are connected together.

Structure Chart of a Game Program



© McGraw Hill LLC

11

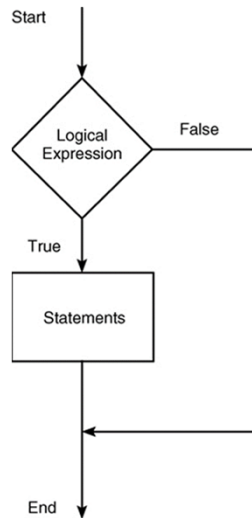
Use of Flowcharts

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.

© McGraw Hill LLC

12

Flowchart representation of the `if` statement



© McGraw Hill LLC

13

Documenting with Pseudocode

We can document with *pseudocode*, in which 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.

© McGraw Hill LLC

14

Finding Bugs

Debugging a program is the process of finding and removing the “bugs,” or errors, in a program. Such errors usually fall into one of the following 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.

Locating Runtime Errors

To locate a runtime error, 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.
3. To test user-defined functions, try commenting out the `function` line and running the file as a script.
4. Use the debugging features of the Editor/Debugger, which is discussed in Section 4.8.

Development of Large Programs

1. Writing and testing of individual modules (the *unit-testing* phase).
2. Writing of the top-level program that uses the modules (the *build* phase). Not all modules are included in the initial testing. As the build proceeds, more modules are included.
3. Testing of the first complete program (the *alpha release* phase). This is usually done only in-house by technical people closely involved with the program development. There might be several alpha releases as bugs are discovered and removed.
4. Testing of the final alpha release by in-house personnel and by familiar and trusted outside users, who often must sign a confidentiality agreement. This is the *beta release* phase, and there might be several beta releases.

Relational Operators ₁

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

Relational Operators ₂

For example, suppose that $x = [6, 3, 9]$ and $y = [14, 2, 9]$. The following MATLAB session shows some examples.

```
>>z = (x < y)
z =
     1     0     0
>>z = (x ~= y)
z =
     1     1     0
>>z = (x > 8)
z =
     0     0     1
```

Relational Operators ₃

The relational operators can be used for array addressing.

For example, with $x = [6, 3, 9]$ and $y = [14, 2, 9]$, typing

```
z = x(x < y)
```

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

Relational Operators ₄

The arithmetic operators $+$, $-$, $*$, $/$, and \backslash 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 ₁

When the relational operators are used, such as

$$x = (5 > 2)$$

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

Logical is a first-class data type and a MATLAB class, and so logical is equivalent to other first-class types such as character and cell arrays.

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

The logical Class ₂

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.
```

© McGraw Hill LLC

23

Accessing Arrays Using Logical Arrays ₁

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.

© McGraw Hill LLC

24

Accessing Arrays Using Logical Arrays ₂

Specifying array subscripts with logical arrays extracts the elements that correspond to the true (1) elements in the logical array.

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

```
>> A=[5,6,7;8,9,10;11,12,13] >> B=logical(eye(3))

A =

     5     6     7
     8     9    10
    11    12    13

>> eye(3)

ans =

     1     0     0
     0     1     0
     0     0     1

>> C=A(B)

C =

     5
     9
    13
```

© McGraw Hill LLC

25

Logical Operators ₁

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.

© McGraw Hill LLC

26

Logical Operators ₂

Operator	Name	Definition
& &	Short-Circuit AND	Operator for scalar logical expressions. A & & B returns true if both A and B evaluate to true, and false if they do not.
	Short-Circuit OR	Operator for scalar logical expressions. A B returns true if either A or B or both evaluate to true, and false if they do not.

Order of Precedence for Operator Types

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 1

Logical function	Definition
<code>all(x)</code>	Returns a scalar, which is 1 if all the elements in the vector <code>x</code> are nonzero and 0 otherwise.
<code>all(A)</code>	Returns a row vector having the same number of columns as the matrix <code>A</code> and containing ones and zeros, depending on whether or not the corresponding column of <code>A</code> has all nonzero elements.
<code>any(x)</code>	Returns a scalar, which is 1 if any of the elements in the vector <code>x</code> is nonzero and 0 otherwise.

```

>> A=[1,2,3;0,0,0;-1,-2,-4]
>> x=[-1,0,2.5,4];
>> y=all(x);
>> y
y =
    logical
    0

>> all(A)
ans =
    1x3 logical array
    0    0    0

>> x=[2,4,0,-1.5];
>> any(x)
ans =
    logical
    1

```

© McGraw Hill LLC

29

Logical Functions 2

Logical function	Definition
<code>any(A)</code>	Returns a row vector having the same number of columns as <code>A</code> and containing ones and zeros, depending on whether or not the corresponding column of the matrix <code>A</code> contains any nonzero elements.
<code>isfinite(A)</code>	Returns an array of the same dimension as <code>A</code> with ones where the elements of <code>A</code> are finite and zeros elsewhere.

```

>> A=[1,2,3;0,0,-1;3,4.5,5]
>> isfinite(A)
ans =
    3x3 logical array
    1    1    1
    1    1    1
    1    1    1

>> any(A)
ans =
    1x3 logical array
    1    1    1

```

© McGraw Hill LLC

30

Logical Functions ₃

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.

```
A =
     1     2     3
    -1    -2     0
```

```
>> ischar(A)
```

```
ans =
```

```
logical
```

```
0
```

```
>> A='Hello students';
```

```
>> ischar(A)
```

```
ans =
```

```
logical
```

```
1
```

```
>> A=[0,0;0,0];      >> C=[];
>> B=[1,0;0,0];      >> isempty(C)
>> isempty(A)
```

```
ans =
```

```
logical
```

```
0
```

```
>> isempty(B)
```

```
ans =
```

```
logical
```

```
0
```

```
logical
```

```
1
```

© McGraw Hill LLC

31

Logical Functions ₄

Logical function Definition

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.)

```
>> A=[1,inf,-2,-3];
>> isinf(A)
```

```
ans =
```

```
1×4 logical array
```

```
0     1     0     0
```

```
>> B=[2,NaN,sqrt(2),inf];
```

```
>> isnan(B)
```

```
ans =
```

```
1×4 logical array
```

```
0     1     0     0
```

© McGraw Hill LLC

32

Logical Functions ₅

`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.

```
>> A=[1,2,3];
>> B='Try this';
>> isnumeric(A)

ans =

    logical

     1

>> isnumeric(B)

ans =

    logical

     0

>> A=[1+2i,-sqrt(3),inf];
>> isreal(A)

ans =

    logical

     0
```

Logical Functions ₆

`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.

```
>> A=[3>5,-2==3,-3~=3];
>> logical(A)

ans =

    1×3 logical array

     0     0     1

>> A=[1,2,3,4,0];
>> B=[0,0,0,1,-2];
>> xor(A,B)

ans =

    1×5 logical array

     1     1     1     0     1
```

The find Function

`find(A)`

Computes an array containing the indices of the nonzero elements of the array A.

`[u,v,w] = find(A)`

Computes the arrays u and v containing the row and column indices of the nonzero elements of the array A and computes the array w containing the values of the nonzero elements. The array w may be omitted.

```
>> A=[-1,0,-sqrt(3),inf,NaN];
>> [u,v,w]=find(A)

u =
     1     1     1     1

v =
     1     3     4     5

w =
    -1.0000    -1.7321     Inf     NaN
```

© McGraw Hill LLC

35

Logical Operators and the find Function

Consider the session

```
>>x = [5, -3, 0, 0, 8];
```

```
>>y = [2, 4, 0, 5, 7];
```

```
>>z = find(x&y)
```

z =

```
     1     2     5
```

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

```
>> x=[5,-3,0,0,8];
>> y=[2,4,0,5,7];
>> B=x&y
```

B =

1x5 **logical** array

```
     1     1     0     0     1
```

```
>> z=find(B)
```

z =

```
     1     2     5
```

```
>> z=find(x&y)
```

z =

```
     1     2     5
```

© McGraw Hill LLC

36

Logical Operators and the find Function 2

In the following session, note the difference between the result obtained by `y(x&y)` and the result obtained by `find(x&y)` in the previous slide.

```
>>x = [5, -3, 0, 0, 8];
```

```
y = [2, 4, 0, 5, 7];
```

```
>>values = y(x&y)
```

```
values =      2      4      7
```

```
>>how_many = length(values)
```

```
how_many = 3
```

```
>> x=[5,-3,0,0,8];      >> y(x&y)
>> y=[2,4,0,5,7];      ans =
>> B=x&y
B =      2      4      7
1×5 logical array      >> length(y(x&y))
1  1  0  0  1      ans =
>> y(B)      3
ans =
      2      4      7
```

© McGraw Hill LLC

37

Question 1

If `x = [-10,14,2,5]` and `y = [-9,13,0,4]`, what will be the result of the `~y > x`.

a) 1 0 0 0 (correct)

b) 1 1 1 1

c) 1 1 1 0

d) 0 1 1 1

```
>> x=[-10,14,2,5];
>> y=[-9,13,0,4];
>> ~y
ans =
1×4 logical array
0  0  1  0
>> x
x =
-10  14  2  5
>> ~y>x
ans =
1×4 logical array
1  0  0  0
```

© McGraw Hill LLC

38

Question 2

Suppose that $x = [-5, 4, 3, -8, 6]$ and $y = [-10, 6, 3, 5, 4]$. What is the result of the $B = (x \sim y)$ operation?

- a) 1 1 1 1 1
- b) 1 1 1 0 0
- c) 1 1 0 1 1 (correct)
- d) 1 1 0 0 1

```
>> x = [-5, 4, 3, -8, 6];
>> y = [-10, 6, 3, 5, 4];
>> B = x ~ y

B =

1x5 logical array

1 1 0 1 1
```

Question 3

Suppose that $x = [-2, 5, 4, 7, 1]$ and $y = [3, 5, -2, 1, 6]$. Use MATLAB to find the indices of the elements in x that are greater than the corresponding elements in y .

- a) 1 2 5
- b) 3 4 (correct)
- c) 2 3
- d) 1 3 5

```
>> x = [-2, 5, 4, 7, 1];
>> y = [3, 5, -2, 1, 6];
>> B = (x > y)

B =

1x5 logical array

0 0 1 1 0

>> Z = x(B)

Z =

4 7

>> T = find(B)

T =

3 4
```

Question 4

The height and speed of a projectile (such as a thrown ball) launched with a speed of v_0 at an angle A to the horizontal are given by

$$h(t) = v_0 t \sin A - 0.5g t^2$$

$$v(t) = \sqrt{v_0^2 - 2v_0 g t \sin A + g^2 t^2}$$

where g is the acceleration due to gravity. The projectile will strike the ground when $h(t) = 0$, which gives the time to hit $t_{hit} = 2(v_0/g)\sin A$. Suppose that $A = 40^\circ$, $v_0 = 20 \text{ m/s}$, and $g = 9.81 \text{ m/s}^2$.

Use the MATLAB relational and logical operators to find the first time when $h \geq 6$.

Hint: When computing the times t_1 and t_2 , you have to keep in mind that the first element in the array t corresponds to $t = 0$ (that is, $t(1) = 0$).

```
>> %Set the initial values.
>> v0=20;g=9.81;A=40*pi/180;
>> %Compute the time to hit
>> t_hit=2*v0*sin(A)/g;
>> %Define arrays containing t,h and v.
>> t=0:t_hit/100:t_hit;

>> h=v0*t*sin(A)-0.5*g*t.^2;
>> v=sqrt(v0^2-2*v0*g*sin(A)*t+g^2*t.^2);
>> %Determine when the height is not less than 6
>> u=find(h>=6);
>> %Compute the corresponding time.
>> t_1=u(1)*(t_hit/100)

t_1 =

0.6552
```

© McGraw Hill LLC

41

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.

© McGraw Hill LLC

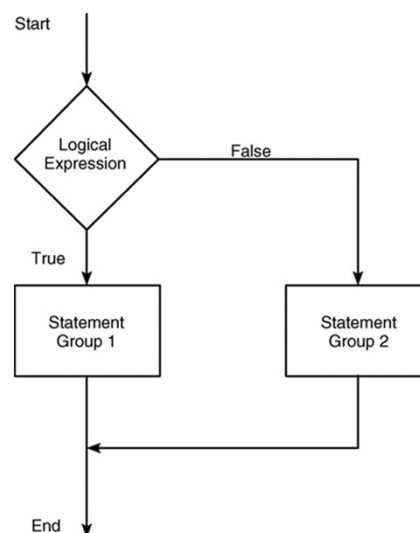
42

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
```

Flowchart of the `else` Structure



The Logical Expression in the `else` Structure ₁

The logical expression may be an *array*. If so, the test returns a value of true only if *all* the elements of the logical expression are true!

The Logical Expression in the `else` Structure ₂

For 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('All elements of x are negative.')
else
    y = sqrt(x)
end
```

When this program is run it gives the result

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

The Logical Expression in the else Structure ₃

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

```
x = [4, -9, 25];
if x >= 0
    y = sqrt(x)
else
    disp('Some of the elements of x are negative.')
end
```

When executed, it produces the following message:

Some of the elements of x are negative.

The test if $x < 0$ is false, and the test if $x \geq 0$ also returns a false value because $x \geq 0$ returns the vector $[1, 0, 1]$.

Making Statements Concise

```
if logical expression 1
    if logical expression 2
        statements
    end
end
```

can be replaced with the more concise program

```
if logical expression 1 & logical expression 2
    statements
end
```

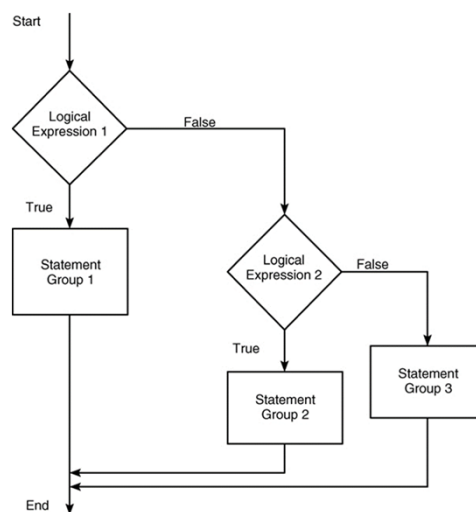

The elseif Statement

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.

Flowchart for the General if-elseif-else Structure



Example of an if-elseif-else Structure

For example, suppose that $y = \log(x)$

for $x > 10$, $y = \sqrt{x}$ for $0 \leq x \leq 10$,

and $y = \exp(x) - 1$ for $x < 0$.

The following statements will compute y if x already has a scalar value.

```
if x > 10
    y = log(x)
elseif x >= 0
    y = sqrt(x)
else
    y = exp(x) - 1
end
```

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 = 'Leslie Student'

name =

    Leslie Student
```

Strings and Conditional Statements ₂

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

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

```
number =
```

```
123
```

Strings and Conditional Statements ₃

The following prompt program uses the `isempty(x)` function, which returns a 1 if the array `x` is empty and 0 otherwise.

It also uses the `input` function, whose syntax is

```
x = input('prompt', 'string')
```

This function displays the string *prompt* on the screen, waits for input from the keyboard, and returns the entered value in the string variable `x`.

The function returns an empty matrix if you press the **Enter** key without typing anything.

Strings and Conditional Statements 4

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('Do you want to continue?
Y/N [Y]: ', 's');

if (isempty(response)) || (response ==
'Y') || (response == 'y')
    response = 'Y'
else
    response = 'N'
end
```

© McGraw Hill LLC

55

Question 5

Q5- Write a script file using conditional statements to evaluate the following function, assuming that the scalar variable x has a value. The function is

$$y = \sqrt{x^2 + 1}$$

$$y = 3x + 1$$

$$y = 9 \sin(5x - 50) + 31$$

$$\text{for } x < 0,$$

$$\text{for } 0 \leq x < 10$$

$$\text{for } x \geq 10.$$

Use your file to evaluate y for $x = 11$, and check the results by hand.

```
>> y=9*sin(5*11-50)+31
```

```
y =
```

```
22.3697
```

```
>> x=input('Enter x:');
Enter x:11
>> if x<0
y=sqrt(x.^2+1)
end
>> if x>=0&x<10
y=3.*x+1
end
>> if x>=10
y=9*sin(5.*x-50)+31
end
```

```
y =
```

```
22.3697
```

© McGraw Hill LLC

56

Question 6

Q6- Create a MATLAB function called fxy to evaluate the function $f(x,y)$ defined as follows:

$$f(x,y) = \begin{cases} xy & \text{if } x \geq 0 \text{ and } y \geq 0 \\ xy^2 & \text{if } x \geq 0 \text{ and } y < 0 \\ x^2y & \text{if } x < 0 \text{ and } y \geq 0 \\ x^2y^2 & \text{if } x < 0 \text{ and } y < 0 \end{cases}$$

For $x=-5$, $y=7$ the value returned by the function $f(x,y)$ is

```

1  function f=fxy(x,y)
2  %evaluate the function:
3  %f=xy if x>=0 and y>=0
4  %f=xy^2 if x>=0 and y<=0
5  %f=x^2y if x<0 and y>=0
6  %f=x^2y^2 if x<0 and y<0
7  if x>=0 & y>=0
8      f=x.*y;
9  elseif x>=0 & y<0
10     f=x.*y.^2;
11 elseif x<0 & y>=0
12     f=x.^2*y;
13 else
14     f=x.^2*y.^2;
15 end
16 end

```

>> fxy(-5,7)

ans =

175

© McGraw Hill LLC

57

Question 7

Q7- Given a number x and the quadrant q ($q = 1, 2, 3, 4$), write a function, func1, to compute $\sin^{-1}(x)$ in degrees, taking into account the quadrant. The program should display an error message if $|x| > 1$. The value returned by func1(-0.6, 3) is

```

function z=func1(x,q)
if abs(x)<1 & x>0
    if q==1
        y = asin(x)*180/pi;
    elseif q==2
        y = 180- asin(x)*180/pi;
    end
elseif (q==3|q==4)
    disp('Incompatible values of x and q')
else
    disp(y)
end
end
if abs(x)<1 & x<0
    if q==4
        y = asin(x)*180/pi;
    elseif q==3
        y = -180-asin(x)*180/pi;
    end
end

```

func1(-0.60, 3)

-143.1301

© McGraw Hill LLC

58

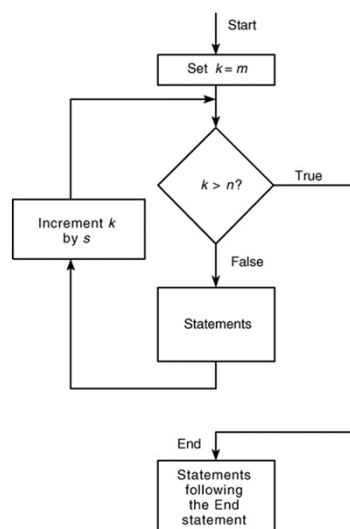
for Loops

A simple example of a `for` loop is

```
for k = 5:10:35
    x = k^2
end
```

The *loop variable* `k` is initially assigned the value 5, and `x` is calculated from $x = k^2$. Each successive pass through the loop increments `k` by 10 and calculates `x` until `k` exceeds 35. Thus, `k` takes on the values 5, 15, 25, and 35, and `x` takes on the values 25, 225, 625, and 1225. The program then continues to execute any statements following the end statement.

Flowchart of a `for` Loop



Loop Variable Expression ₁

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 one.
- 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.

Loop Variable Expression ₂

For example, 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 $y = 1, 3, \text{NaN}, 2$.

Using a Logical Array as a Mask

We can often avoid the use of loops and branching and thus create simpler and faster programs by using a logical array as a *mask* that selects elements of another array. Any elements not selected will remain unchanged.

The following session creates the logical array C from the numeric array A given previously.

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

The result is

$$C = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Example of the Mask Technique

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

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


Question 8

Write a script file to compute the sum of the first 15 terms in the series, $5k^2 - 2k$, $k = 1, 2, 3, \dots, 15$.

- a) 5960 (correct)
- b) 1815
- c) 6643
- d) 3983

```
>> total=0;
>> for k=1:15
    total=5*k^2-2*k+total;
end
>> disp('The sum for the 15 terms is:')
The sum for the 15 terms is:
>> disp(total)
5960
```

Question 9

Write a program to produce the following matrix:

$$A = \begin{bmatrix} 4 & 8 & 12 \\ 10 & 14 & 18 \\ 16 & 20 & 24 \\ 22 & 26 & 30 \end{bmatrix}$$

Then calculate $y = A^2$, the value of $y(3,2)$ is

- a) 400 (correct)
- b) 576
- c) 600
- d) 900

```
matrix.m x +
1 function [z]=matrix
2 for c=1:3
3     for r=1:4
4         A(r,c)=4+(c-1)*4+(r-1)*6;
5     end
6 end
7 z=A;
8 end
```

```
>> y=ans.^2
y =
    4    8    12    16    64   144
   10   14   18   100   196   324
   16   20   24   256   400   576
   22   26   30   484   676   900

>> y(3,2)
ans =
    400
```

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, and 17.

© McGraw Hill LLC

67

The Typical Structure of a while Loop

```
while logical expression
    statements
end
```

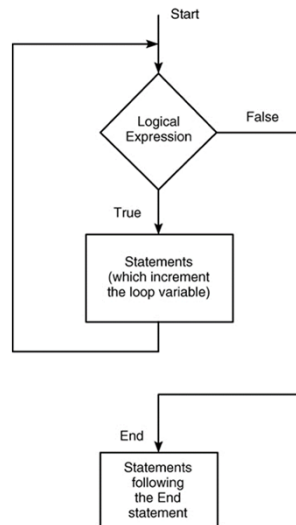
For the `while` loop to function properly, the following two conditions must occur:

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*.

© McGraw Hill LLC

68

Flowchart of the while Loop



© McGraw Hill LLC

69

Question 10

Use a while loop to determine how many terms in the series $3k^2$, $k = 1, 2, 3, \dots$, are required for the sum of the terms to exceed 5620. What is the sum for this number of terms?

- a) 25, 9327
- b) 21, 7753
- c) 19, 6327 (correct)
- d) 17, 4315

```

>> sum=0;
>> k=1;
>> while sum<5620;
sum=sum+3*k^2;
k=k+1;
end
>> disp('sum=')
sum=
>> disp(sum)
6327

>> disp(k)
19

>> sum

sum =

6327
  
```

© McGraw Hill LLC

70

Question 11

Use a while loop to find two decimal places the largest value of x before the error in the $e^x \approx 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$ exceeds 15 percent.

- a) 0.89
- b) 2.04 (correct)
- c) 1.75
- d) 3.21

```
>> error=0;
>> x=0;
>> while error<15
x=x+0.01;
approx=1+x+x^2/2+x^3/6;
error=100*(exp(x)-approx)/exp(x);
end
>> x

x =

    2.0400
```

The switch Structure

The switch structure provides an alternative to using the if, elseif, and else commands. Anything programmed using switch can also be programmed using if structures.

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

Syntax of the switch Structure

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

Example of the switch Structure

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
```



Because learning changes everything.®

www.mheducation.com

© McGraw Hill LLC. All rights reserved. No reproduction or distribution without the prior written consent of McGraw Hill LLC.