Chapter 6
Programming in MATLAB

A computer program is a sequence of computer commands. In a simple program the commands are executed one after the other in the order they are typed. In this book, for example, all the programs that have been presented so far in script files are simple programs. Many situations, however, require more sophisticated programs in which commands are not necessarily executed in the order they are typed, or different commands (or groups of commands) are executed when the program runs with different input variables. For example, a computer program that calculates the cost of mailing a package uses different mathematical expressions to calculate the cost depending on the weight and size of the package, the content (books are less expensive to mail), and the type of service (airmail, ground, etc.). In other situations there might be a need to repeat a sequence of commands several times within a program. For example, programs that solve equations numerically repeat a sequence of calculations until the error in the answer is smaller than some measure.

MATLAB provides several tools that can be used to control the flow of a program. Conditional statements (Section 6.2) and the `switch` structure (Section 6.3) make it possible to skip commands or to execute specific groups of commands in different situations. `For` loops and `while` loops (Section 6.4) make it possible to repeat a sequence of commands several times.

It is obvious that changing the flow of a program requires some kind of decision-making process within the program. The computer must decide whether to execute the next command or to skip one or more commands and continue at a different line in the program. The program makes these decisions by comparing values of variables. This is done by using relational and logical operators, which are explained in Section 6.1.

It should also be noted that user-defined functions (introduced in Chapter 7) can be used in programming. A user-defined function can be used as a subprogram. When the main program reaches the command line that has the user-defined function, it provides input to the function and "waits" for the results. The user-

www.it-ebooks.info

defined function carries out the calculations and transfers the results back to the main program, which then continues to the next command.

6.1 Relational and Logical Operators

A relational operator compares two numbers by determining whether a comparison statement (e.g., 5 < 8) is true or false. If the statement is true, it is assigned a value of 1. If the statement is false, it is assigned a value of 0. A logical operator examines true/false statements and produces a result that is true (1) or false (0) according to the specific operator. For example, the logical AND operator gives 1 only if both statements are true. Relational and logical operators can be used in mathematical expressions and, as will be shown in this chapter, are frequently used in combination with other commands to make decisions that control the flow of a computer program.

Relational operators:

Relational operators in MATLAB are:

Relational operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
~=	Not Equal to

Note that the "equal to" relational operator consists of two = signs (with no space between them), since one = sign is the assignment operator. In other relational operators that consist of two characters, there also is no space between the characters (<=, >=, ~=).

- Relational operators are used as arithmetic operators within a mathematical expression. The result can be used in other mathematical operations, in addressing arrays, and together with other MATLAB commands (e.g., `if`) to control the flow of a program.
- When two numbers are compared, the result is 1 (logical true) if the comparison, according to the relational operator, is true, and 0 (logical false) if the comparison is false.
- If two scalars are compared, the result is a scalar 1 or 0. If two arrays are compared (only arrays of the same size can be compared), the comparison is done *element-by-element*, and the result is a logical array of the same size with 1s and 0s according to the outcome of the comparison at each address.

www.it-ebooks.info

- If a scalar is compared with an array, the scalar is compared with every element of the array, and the result is a logical array with 1s and 0s according to the outcome of the comparison of each element.

Some examples are:

```
>> 5>8                                   Checks if 5 is larger than 8.
ans =                                    Since the comparison is false (5 is
     0                                   not larger than 8) the answer is 0.
>> a=5<10          Checks if 5 is smaller than 10, and assigns the answer to a.
a =                                      Since the comparison is true (5 is smaller
     1                                   than 10) the number 1 is assigned to a.
                                         Using relational operators
>> y=(6<10)+(7>8)+(5*3==60/4)            in math expression.
```

Equal to 1 since 6 is smaller than 10.

Equal to 0 since 7 is not larger than 8.

Equal to 1 since 5*3 is equal to 60/4.

```
y =
     2                                   Define vectors b and c.
>> b=[15 6 9 4 11 7 14]; c=[8 20 9 2 19 7 10];
>> d=c>=b   Checks which c elements are larger than or equal to b elements.
d =
     0     1     1     0     1     1     0
   Assigns 1 where an element of c is larger than or equal to an element of b.
>> b == c              Checks which b elements are equal to c elements.
ans =
     0     0     1     0     0     1     0
>> b~=c               Checks which b elements are not equal to c elements.
ans =
     1     1     0     1     1     0     1
>> f=b-c>0                              Subtracts c from b and then checks
f =                                     which elements are larger than zero.
     1     0     0     1     0     0     1
>> A=[2 9 4; -3 5 2; 6 7 -1]            Define a 3 × 3 matrix A.
A =
     2     9     4
    -3     5     2
     6     7    -1
                          Checks which elements in A are smaller than
                          or equal to 2. Assigns the results to matrix B.
>> B=A<=2
```

www.it-ebooks.info

```
B =
     1     0     0
     1     0     1
     0     0     1
```

- The results of a relational operation with vectors, which are vectors with 0s and 1s, are called logical vectors and can be used for addressing vectors. When a logical vector is used for addressing another vector, it extracts from that vector the elements in the positions where the logical vector has 1s. For example:

```
>> r = [8 12 9 4 23 19 10]                    Define a vector r.
r =
     8    12     9     4    23    19    10
>> s=r<=10        Checks which r elements are smaller than or equal to 10.
s =
     1     0     1     1     0     0     1
                        A logical vector s with 1s at positions where
                        elements of r are smaller than or equal to 10.
>> t=r(s)                Use s for addresses in vector r to create vector t.
t =                                  Vector t consists of elements of
     8     9     4    10               r in positions where s has 1s.
>> w=r(r<=10)                  The same procedure can be done in one step.
w =
     8     9     4    10
```

- Numerical vectors and arrays with the numbers 0s and 1s are not the same as logical vectors and arrays with 0s and 1s. Numerical vectors and arrays can not be used for addressing. Logical vectors and arrays, however, can be used in arithmetic operations. The first time a logical vector or an array is used in arithmetic operations it is changed to a numerical vector or array.
- Order of precedence: In a mathematical expression that includes relational and arithmetic operations, the arithmetic operations (+, –, *, /, \) have precedence over relational operations. The relational operators themselves have equal precedence and are evaluated from left to right. Parentheses can be used to alter the order of precedence. Examples are:

```
>> 3+4<16/2                              + and / are executed first.
ans =                                The answer is 1 since 7 < 8 is true.
     1
>> 3+(4<16)/2     4 < 16 is executed first, and is equal to 1, since it is true.
ans =                                   3.5 is obtained from 3 + 1/2.
    3.5000
```

www.it-ebooks.info

Logical operators:

Logical operators in MATLAB are:

Logical operator	Name	Description
& Example: A&B	AND	Operates on two operands (A and B). If both are true, the result is true (1); otherwise the result is false (0).
| Example: A|B	OR	Operates on two operands (A and B). If either one, or both, are true, the result is true (1); otherwise (both are false) the result is false (0).
~ Example: ~A	NOT	Operates on one operand (A). Gives the opposite of the operand; true (1) if the operand is false, and false (0) if the operand is true.

- Logical operators have numbers as operands. A nonzero number is true, and a zero number is false.
- Logical operators (like relational operators) are used as arithmetic operators within a mathematical expression. The result can be used in other mathematical operations, in addressing arrays, and together with other MATLAB commands (e.g., `if`) to control the flow of a program.
- Logical operators (like relational operators) can be used with scalars and arrays.
- The logical operations AND and OR can have both operands as scalars, both as arrays, or one as an array and one as a scalar. If both are scalars, the result is a scalar 0 or 1. If both are arrays, they must be of the same size and the logical operation is done *element-by-element*. The result is an array of the same size with 1s and 0s according to the outcome of the operation at each position. If one operand is a scalar and the other is an array, the logical operation is done between the scalar and each of the elements in the array and the outcome is an array of the same size with 1s and 0s.
- The logical operation NOT has one operand. When it is used with a scalar, the outcome is a scalar 0 or 1. When it is used with an array, the outcome is an array of the same size with 0s in positions where the array has nonzero numbers and 1s in positions where the array has 0s.

Following are some examples:

```
>> 3&7
```

3 AND 7.

www.it-ebooks.info

```
ans =                                   3 and 7 are both true (nonzero), so the outcome is 1.
     1
>> a=5|0                                5 OR 0 (assign to variable a).
a =                          1 is assigned to a since at least one number is true (nonzero).
     1
>> ~25                                  NOT 25.
ans =                                   The outcome is 0 since 25 is true
     0                                  (nonzero) and the opposite is false.
>> t=25*((12&0)+(~0)+(0|5))             Using logical operators in a math expression.
t =
    50
>> x=[9 3 0 11 0 15]; y=[2 0 13 -11 0 4];   Define two vectors x and y.
>> x&y                                  The outcome is a vector with 1 in every position where
ans =                                   both x and y are true (nonzero elements), and 0s otherwise.
     1     0     0     1     0     1
>> z=x|y                                The outcome is a vector with 1 in every position where either
z =                                     or both x and y are true (nonzero elements), and 0s otherwise.
     1     1     1     1     0     1
                                        The outcome is a vector with 0 in every position where
>> ~(x+y)                               the vector x + y is true (nonzero elements), and 1 in
ans =                                   every position where x + y is false (zero elements).
     0     0     0     1     1     0
```

Order of precedence:

Arithmetic, relational, and logical operators can be combined in mathematical expressions. When an expression has such a combination, the result depends on the order in which the operations are carried out. The following is the order used by MATLAB:

Precedence	Operation
1 (highest)	Parentheses (if nested parentheses exist, inner ones have precedence)
2	Exponentiation
3	Logical NOT (~)
4	Multiplication, division
5	Addition, subtraction
6	Relational operators (>, <, >=, <=, ==, ~=)
7	Logical AND (&)
8 (lowest)	Logical OR (|)

www.it-ebooks.info

If two or more operations have the same precedence, the expression is executed in order from left to right.

It should be pointed out here that the order shown above is the one used since MATLAB 6. Previous versions of MATLAB used a slightly different order (& did not have precedence over |), so the user must be careful. Compatibility problems between different versions of MATLAB can be avoided by using parentheses even when they are not required.

The following are examples of expressions that include arithmetic, relational, and logical operators:

```
>> x=-2; y=5;
```
Define variables x and y.

```
>> -5<x<-1
ans =
     0
```
This inequality is correct mathematically. The answer, however, is false since MATLAB executes from left to right. $-5 < x$ is true (=1) and then $1 < -1$ is false (0).

```
>> -5<x & x<-1
ans =
     1
```
The mathematically correct statement is obtained by using the logical operator &. The inequalities are executed first. Since both are true (1), the answer is 1.

```
>> ~(y<7)
ans =
     0
```
$y < 7$ is executed first, it is true (1), and ~1 is 0.

```
>> ~y<7
ans =
     1
```
~y is executed first, y is true (1) (since y is nonzero), ~1 is 0, and $0 < 7$ is true (1).

```
>> ~((y>=8)|(x<-1))
ans =
     0
```
$y >= 8$ (false), and $x < -1$ (true) are executed first. OR is executed next (true). ~ is executed last, and gives false (0).

```
>> ~(y>=8)|(x<-1)
ans =
     1
```
$y >= 8$ (false), and $x < -1$ (true) are executed first. NOT of ($y >= 8$) is executed next (true). OR is executed last, and gives true (1).

Built-in logical functions:

MATLAB has built-in functions that are equivalent to the logical operators. These functions are:

`and(A,B)`	equivalent to `A&B`
`or(A,B)`	equivalent to `A|B`
`not(A)`	equivalent to `~A`

www.it-ebooks.info

In addition, MATLAB has other logical built-in functions, some of which are described in the following table:

Function	Description	Example

`xor(a,b)`	Exclusive or. Returns true (1) if one operand is true and the other is false.	`>> xor(7,0)`
		`ans =`
		`1`
		`>> xor(7,-5)`
		`ans =`
		`0`

`all(A)`	Returns 1 (true) if all elements in a vector `A` are true (nonzero). Returns 0 (false) if one or more elements are false (zero).
	If `A` is a matrix, treats columns of `A` as vectors, and returns a vector with 1s and 0s.
		`>> A=[6 2 15 9 7 11];`
		`>> all(A)`
		`ans =`
		`1`
		`>> B=[6 2 15 9 0 11];`
		`>> all(B)`
		`ans =`
		`0`

`any(A)`	Returns 1 (true) if any element in a vector `A` is true (nonzero). Returns 0 (false) if all elements are false (zero).
	If `A` is a matrix, treats columns of `A` as vectors, and returns a vector with 1s and 0s.
		`>> A=[6 0 15 0 0 11];`
		`>> any(A)`
		`ans =`
		`1`
		`>> B = [0 0 0 0 0 0];`
		`>> any(B)`
		`ans =`
		`0`

`find(A)`
 	
`find(A>d)`
	If `A` is a vector, returns the indices of the nonzero elements.
	If `A` is a vector, returns the address of the elements that are larger than `d` (any relational operator can be used).
		`>> A=[0 9 4 3 7 0 0 1 8];`
		`>> find(A)`
		`ans =`
		`2 3 4 5 8 9`
		`>> find(A>4)`
		`ans =`
		`2 5 9`

www.it-ebooks.info

The operations of the four logical operators, `and`, `or`, `xor`, and `not` can be summarized in a truth table:

INPUT		OUTPUT				
A	B	AND A&B	OR A|B	XOR (A,B)	NOT ~A	NOT ~B
false	false	false	false	false	true	true
false	true	false	true	true	true	false
true	false	false	true	true	false	true
true	true	true	true	false	false	false

Sample Problem 6-1: Analysis of temperature data

The following were the daily maximum temperatures (in °F) in Washington, DC, during the month of April 2002: 58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 91 93 89 91 80 59 69 56 64 63 66 64 74 63 69 (data from the U.S. National Oceanic and Atmospheric Administration). Use relational and logical operations to determine the following:

(*a*) The number of days the temperature was above 75°.

(*b*) The number of days the temperature was between 65° and 80°.

(*c*) The days of the month when the temperature was between 50° and 60°.

Solution

In the script file below the temperatures are entered in a vector. Relational and logical expressions are then used to analyze the data.

```
T=[58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 ...
   91 93 89 91 80 59 69 56 64 63 66 64 74 63 69];
Tabove75=T>=75;
```

A vector with 1s at addresses where `T` >= 75.

```
NdaysTabove75=sum(Tabove75)
```

Add all the 1s in the vector `Tabove75`.

```
Tbetween65and80=(T>=65)&(T<=80);
```

A vector with 1s at addresses where `T` >= 65 and `T` <= 80.

```
NdaysTbetween65and80=sum(Tbetween65and80)
```

Add all the 1s in the vector `Tbetween65and80`.

```
datesTbetween50and60=find((T>=50)&(T<=60))
```

The function `find` returns the address of the elements in `T` that have values between 50 and 60.

www.it-ebooks.info

The script file (saved as Exp6_1) is executed in the Command Window:

```
>> Exp6_1
NdaysTabove75 =                                  For 7 days the temp was above 75.
     7
NdaysTbetween65and80 =    For 12 days the temp was between 65 and 80.
    12
datesTbetween50and60 =                          Dates of the month with
     1     4     5     7    21    23             temp between 50 and 60.
```

6.2 CONDITIONAL STATEMENTS

A conditional statement is a command that allows MATLAB to make a decision of whether to execute a group of commands that follow the conditional statement, or to skip these commands. In a conditional statement, a conditional expression is stated. If the expression is true, a group of commands that follow the statement are executed. If the expression is false, the computer skips the group. The basic form of a conditional statement is:

```
if  conditional expression consisting of relational and/or logical operators.
```

Examples:

```
if a < b
if c >= 5
if a == b
if a ~= 0
if (d<h)&(x>7)
if (x~=13)|(y<0)
```

All the variables must have assigned values.

- Conditional statements can be a part of a program written in a script file or a user-defined function (Chapter 7).
- As shown below, for every `if` statement there is an `end` statement.

The `if` statement is commonly used in three structures, `if-end`, `if-else-end`, and `if-elseif-else-end`, which are described next.

6.2.1 The `if-end` Structure

The `if-end` conditional statement is shown schematically in Figure 6-1. The figure shows how the commands are typed in the program, and a flowchart that symbolically shows the flow, or the sequence, in which the commands are executed. As the program executes, it reaches the `if` statement. If the conditional expres-

www.it-ebooks.info

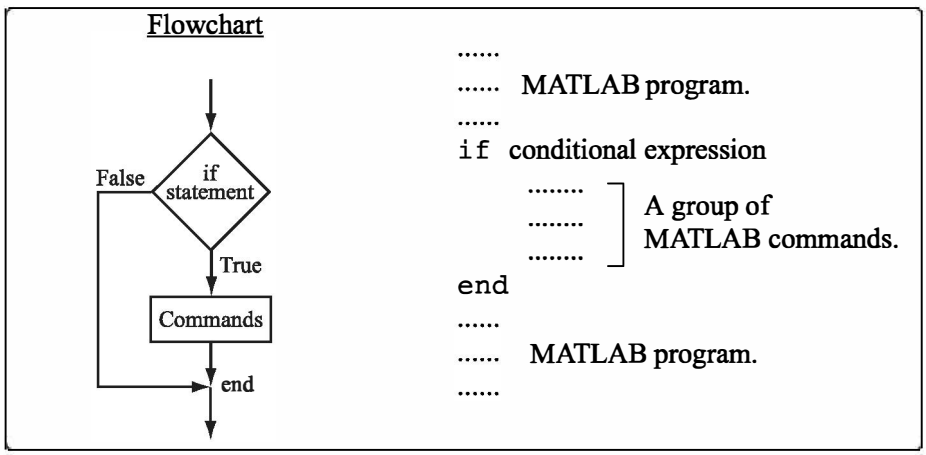


Figure 6-1: The structure of the `if-end` conditional statement.

sion in the `if` statement is true (1), the program continues to execute the commands that follow the `if` statement all the way down to the `end` statement. If the conditional expression is false (0), the program skips the group of commands between the `if` and the `end`, and continues with the commands that follow the `end`.

The words `if` and `end` appear on the screen in blue, and the commands between the `if` statement and the `end` statement are automatically indented (they don't have to be), which makes the program easier to read. An example where the `if-end` statement is used in a script file is shown in Sample Problem 6-2.

Sample Problem 6-2: Calculating worker's pay

A worker is paid according to his hourly wage up to 40 hours, and 50% more for overtime. Write a program in a script file that calculates the pay to a worker. The program asks the user to enter the number of hours and the hourly wage. The program then displays the pay.

Solution

The program in a script file is shown below. The program first calculates the pay by multiplying the number of hours by the hourly wage. Then an `if` statement checks whether the number of hours is greater than 40. If so, the next line is executed and the extra pay for the hours above 40 is added. If not, the program skips to the `end`.

```
t=input('Please enter the number of hours worked  ');
h=input('Please enter the hourly wage in $  ');
Pay=t*h;
if t>40
```

www.it-ebooks.info

```
    Pay=Pay+(t-40)*0.5*h;
end
fprintf('The worker''s pay is  $ %5.2f',Pay)
```

Application of the program (in the Command Window) for two cases is shown below (the file was saved as Workerpay):

```
>> Workerpay
Please enter the number of hours worked  35
Please enter the hourly wage in $  8
The worker's pay is  $ 280.00
>> Workerpay
Please enter the number of hours worked  50
Please enter the hourly wage in $  10
The worker's pay is  $ 550.00
```

6.2.2 *The* `if-else-end` *Structure*

The `if-else-end` structure provides a means for choosing one group of commands, out of a possible two groups, for execution. The `if-else-end` structure is shown in Figure 6-2. The figure shows how the commands are typed in the program, and includes a flowchart that illustrates the flow, or the sequence, in

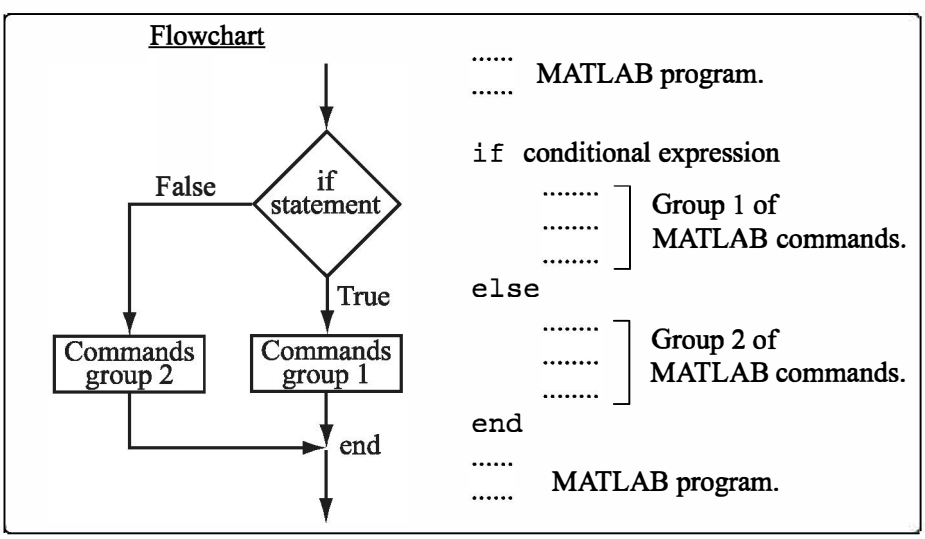


Figure 6-2: The structure of the `if-else-end` conditional statement.

www.it-ebooks.info

which the commands are executed. The first line is an `if` statement with a conditional expression. If the conditional expression is true, the program executes group 1 of commands between the `if` and the `else` statements and then skips to the `end`. If the conditional expression is false, the program skips to the `else` and then executes group 2 of commands between the `else` and the `end`.

6.2.3 The `if-elseif-else-end` *Structure*

The `if-elseif-else-end` structure is shown in Figure 6-3. The figure shows how the commands are typed in the program, and gives a flowchart that illustrates the flow, or the sequence, in which the commands are executed. This structure includes two conditional statements (`if` and `elseif`) that make it possible to select one out of three groups of commands for execution. The first line is an `if` statement with a conditional expression. If the conditional expression is true, the program executes group 1 of commands between the `if` and the

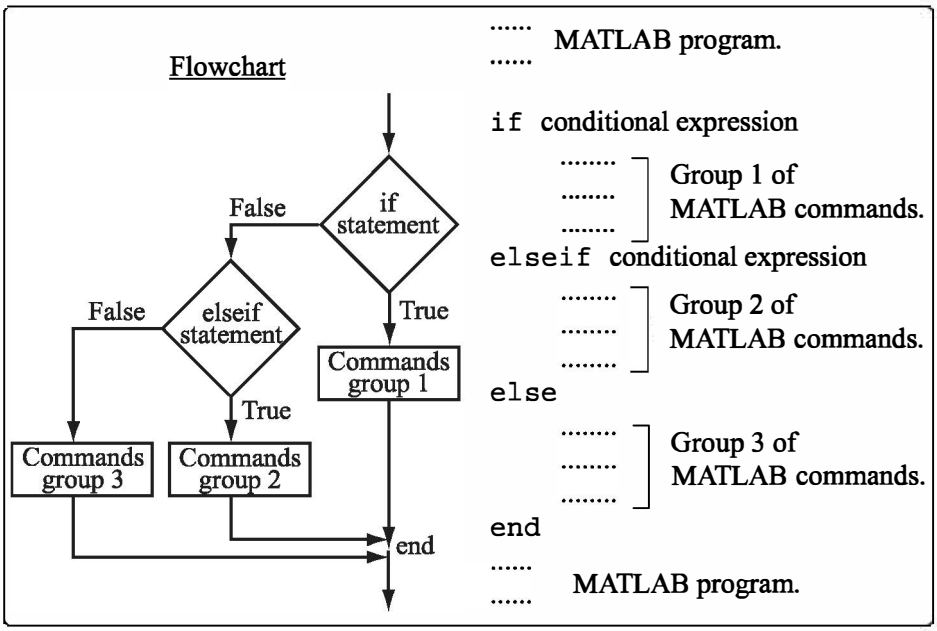


Figure 6-3: The structure of the `if-elseif-else-end` conditional statement.

`elseif` statements and then skips to the `end`. If the conditional expression in the `if` statement is false, the program skips to the `elseif` statement. If the conditional expression in the `elseif` statement is true, the program executes group 2 of commands between the `elseif` and the `else` and then skips to the `end`. If the conditional expression in the `elseif` statement is false, the program skips to the `else` and executes group 3 of commands between the `else` and the `end`.

It should be pointed out here that several `elseif` statements and associ-

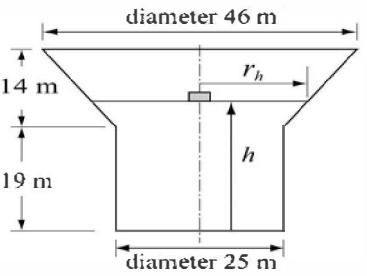
www.it-ebooks.info

ated groups of commands can be added. In this way more conditions can be included. Also, the `else` statement is optional. This means that in the case of several `elseif` statements and no `else` statement, if any of the conditional statements is true the associated commands are executed; otherwise nothing is executed.

The following example uses the `if-elseif-else-end` structure in a program.

Sample Problem 6-3: Water level in water tower

The tank in a water tower has the geometry shown in the figure (the lower part is a cylinder and the upper part is an inverted frustum of a cone). Inside the tank there is a float that indicates the level of the water. Write a MATLAB program that determines the volume of the water in the tank from the position (height h) of the float. The program asks the user to enter a value of h in m, and as output displays the volume of the water in m^3.



Solution

For $0 \le h \le 19$ m the volume of the water is given by the volume of a cylinder with height h: $V = \pi 12.5^2 h$.

For $19 < h \le 33$ m the volume of the water is given by adding the volume of a cylinder with $h = 19$ m, and the volume of the water in the cone:

$$V = \pi 12.5^2 \cdot 19 + \frac{1}{3}\pi(h-19)(12.5^2 + 12.5 \cdot r_h + r_h^2)$$

where $r_h = 12.5 + \frac{10.5}{14}(h-19)$.

The program is:

```
% The program calculates the volume of the water in the
water tower.
h=input('Please enter the height of the float in meter  ');
if h > 33
   disp('ERROR. The height cannot be larger than 33 m.')
elseif h < 0
   disp('ERROR. The height cannot be a negative number.')
elseif h <= 19
   v = pi*12.5^2*h;
   fprintf('The volume of the water is %7.3f cubic meter.\n',v)
```

www.it-ebooks.info

```
else
   rh=12.5+10.5*(h-19)/14;
   v=pi*12.5^2*19+pi*(h-19)*(12.5^2+12.5*rh+rh^2)/3;
   fprintf('The volume of the water is %7.3f cubic meter.\n',v)
end
```

The following is the display in the Command Window when the program is used with three different values of water height.

```
Please enter the height of the float in meter  8
The volume of the water is 3926.991 cubic meter.
Please enter the height of the float in meter  25.7
The volume of the water is 14114.742 cubic meter.
Please enter the height of the float in meter  35
ERROR. The height cannot be larger than 33 m.
```

6.3 THE switch-case STATEMENT

The switch-case statement is another method that can be used to direct the flow of a program. It provides a means for choosing one group of commands for execution out of several possible groups. The structure of the statement is shown in Figure 6-4.

- The first line is the switch command, which has the form:

switch switch expression

The switch expression can be a scalar or a string. Usually it is a variable that has an assigned scalar or a string. It can also be, however, a mathematical expression that includes pre-assigned variables and can be evaluated.

- Following the switch command are one or several case commands. Each has a value (can be a scalar or a string) next to it (value1, value2, etc.) and an associated group of commands below it.
- After the last case command there is an optional otherwise command followed by a group of commands.
- The last line must be an end statement.

How does the switch-case statement work?

The value of the switch expression in the switch command is compared with the values that are next to each of the case statements. If a match is found, the group of commands that follow the case statement with the match are executed. (Only one group of commands—the one between the case that matches and either the

www.it-ebooks.info