The user-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.
- 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
ans =
     0
>> a=5<10
a =
     1

>> y=(6<10)+(7>8)+(5*3==60/4)
y =
     2
>> b=[15 6 9 4 11 7 14]; c=[8 20 9 2 19 7 10];
>> d=c>=b
d =
     0     1     1     0     1     1     0
>> b == c
ans =
     0     0     1     0     0     1     0
>> b~=c
ans =
     1     1     0     1     1     0     1
>> f=b-c>0
f =
     1     0     0     1     0     0     1
>> A=[2 9 4; -3 5 2; 6 7 -1]
A =
     2     9     4
    -3     5     2
     6     7    -1
>> B=A<=2
```

Checks if 5 is larger than 8.

Since the comparison is false (5 is not larger than 8) the answer is 0.

Checks if 5 is smaller than 10, and assigns the answer to a.

Since the comparison is true (5 is smaller than 10) the number 1 is assigned to a.

Using relational operators 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.

Define vectors b and c.

Checks which c elements are larger than or equal to b elements.

Assigns 1 where an element of c is larger than or equal to an element of b.

Checks which b elements are equal to c elements.

Checks which b elements are not equal to c elements.

Subtracts c from b and then checks which elements are larger than zero.

Define a 3×3 matrix A.

Checks which elements in A are smaller than or equal to 2. Assigns the results to matrix B.

```
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
```

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.
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                                                Define two vec-
>> x=[9 3 0 11 0 15]; y=[2 0 13 -11 0 4];           tors 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
                          the vector x + y is true (nonzero elements), and 1 in
>> ~(x+y)                 every position where x + y is false (zero elements).
ans =
     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 (|)

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`

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`

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;
NdaysTabove75=sum(Tabove75)
Tbetween65and80=(T>=65)&(T<=80);
NdaysTbetween65and80=sum(Tbetween65and80)
datesTbetween50and60=find((T>=50)&(T<=60))
```

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

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

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

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

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