Flow Control in MATLAB

1 Overview

Flow control allows computer codes to operate under circumstances with variable inputs and parameter ranges. In short, flow control allows the code to “make choices” during execution. The basic tool for flow control is the `if` construct

1. Plain `if`
2. `if...else`
3. `if...elseif`

An `if` construct is used to isolate a block (or blocks) of code that is executed only if a condition is true.

```
if expression
    block of statements
end
```

2 `if` constructs

2.1 Plain `if`

```
if test expression
  statements to execute when test expression is true
end
```

Example:

```
x = ...
if x<0
  disp('Warning: x is negative');
end
```

2.2 `if...else`

```
if test expression
  block to execute when test expression is true
else
  alternative block to execute when test expression is false
end
```

Example:

```
F = ...
% --- specify beam depth based on applied load
if F<100
  b = 3.5;
else
  b = 6;
end
```

2.3 `if...elseif...else...`

```
if test expression 1
  block to execute when test expression 1 is true
elseif test expression 2
  block to execute when test expression 2 is true
else
  alternative block to execute when both test expression are false
end
```

Example:

```
F = ...
% --- specify beam depth based on applied load
if F<100
  b = 3.5;
elseif F<200
  b = 6;
else
  error('Applied load exceeds 200 lb maximum');
end
```

Warning: It almost always a good idea to include an `else` clause in an `if`. . . `elseif`. . . construct.

3 Logical Expressions and Flow Control

Logical expressions are formulas that have only two possible outcomes: true or false. In MATLAB "false" is the same as the numerical value zero. Any non-zero value is considered to be "true". MATLAB also has the built-in values `true` and `false`

```
>> true
ans =
     1

>> false
ans =
     0
```

3.1 Relational Operators

Relational operators are used in comparisons and yield a logical value.

Operator	Meaning
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`~=`	not equal to

Relational operators can be used on the right hand side of an assignment, as in this concocted example.

```
a = ...
b = ...

a_is_smaller = a<b;
```

The value stored in `a_is_smaller` is `true` if $a < b$, otherwise it is false.

Usually, relational operators are used in the logical expressions in an "`if`" construct.

```
a = ...
b = ...

if a<b
  ...
end
```

3.2 Basic Numerical Comparison

```
if x>tol
  ...
end
```

```
if x<=xmax
  ...
end
```

```
if abs(x-xmax) < delta
  ...
end
```

Warning: It is usually a bad idea to test numerical values for exact equality

```
function tanTest(theta)

a = 3*pi/2;
t = tan(theta);
s = sin(theta);
c = cos(theta);
t2 = sin(theta)/cos(theta);
if t==t2
  fprintf('Tangent calculations are equal for theta = %8.3f rad\n',theta);
else
  fprintf('Tangent calculations are not equal for theta = %8.3f rad\n',theta);
  fprintf('   t = %14.12f    t2 = %14.12f   diff = %12.3e\n',t,t2,t2-t);
end
```

For some angles, `tanTest` shows that the two tangent calculations are equivalent. For other angles, they are not.

```
t = 8*pi*rand(33,1);
for i=1:length(t)
  tanTest(t(i));
end
```

The lesson is that the test for exact equality is susceptible to small round-off errors.

3.3 Compound Expressions

Compound logical expressions can be created using *logical operators*

Operator	Meaning
`&&`	and
`||`	or
`~`	not

Example:

```
x = ...
% --- Is x in the range xmin <= x <= xmax
if x>=xmin && x<=xmax
  ... do something with x
else
  error('x is out of range');
end
```

A compound expression is equivalent to nested `if`

```
x = ...
% --- Stop if x is outside of the range xmin <= x <= xmax
if x>xmax
  if x<xmin
      error('x is out of range');
  end
end
% continue with calculations involving x
   ...
```

In this example, the compound expression conveys the meaning of the logic more directly. *Always choose the code that is easier to understand.*

4 String Comparisons

Strings are not usually of central concern in numerical computing. Nonetheless, there are a few situations where being able to manipulate strings can enhance the usability of a numerical code. The most common string comparisons involve testing for equality with one of the following built-in functions.

`strcmp`	determine whether two strings are identical
`strcmpi`	determine whether two strings are identical while ignoring the case (upper or lower) of the characters in the string.
`strncmp`	determines whether the first n characters of two strings are identical
`strncmpi`	determine whether the first n characters of two strings are identical while ignoring the case (upper or lower) of the characters in the string.

Example:

```
units = 'SI';

if strcmpi(units,'SI')
  ... perform calculation assuming SI units
elseif strcmpi(units,'EE')
  ... convert quantities in EE units to equivalent
      values in SI units
else
  error('"units" variable not set correctly')'
end
```