FF505

Computational Science

Control Flow

Marco Chiarandini (marco@imada.sdu.dk)

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark

Outline

Algorithms and Control Structures

Algorithm: an ordered sequence of instructions that perform some task in a finite amount of time.

Individual statements, instructions or function calls can be numbered and executed in sequence, but an algorithm has the ability to alter the order of its instructions. The order is referred to as control flow.

Three categories of control flow:

- Sequential operations
- Conditional operations: logical conditions that determine actions.
- Iterative operations (loops)

For an imperative or a declarative program a control flow statement is a statement whose execution results in a choice being made as to which of two or more paths should be followed.

For non-strict functional languages (like Matlab), functions and language constructs exist to achieve the same result, but they are not necessarily called control flow statements (eg, vectorization).

Relational Operators

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

```
islogical(5~=8)
ans =
     1
islogical(logical(5+8))
ans =

     1
>> logical(5+8)
ans =
     1
>> double(6>8)
ans =
     0
>> isnumeric(double(6>8))
ans =
     1
```

Logical Operators

`~`	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	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.

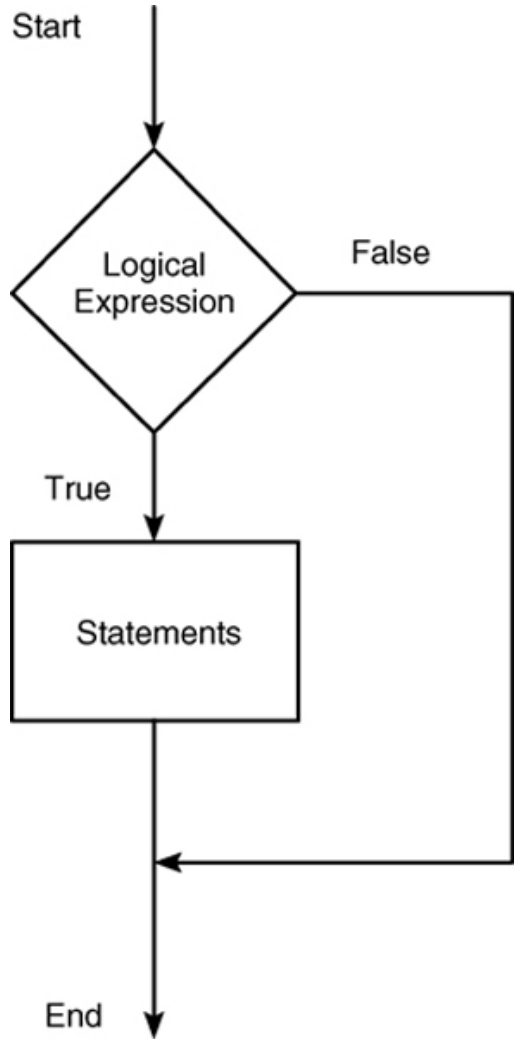
Precedence

1. Parentheses; evaluated starting with the innermost pair.
2. Arithmetic operators and logical NOT (~); evaluated from left to right.
3. Relational operators; evaluated from left to right.
4. Logical AND.
5. Logical OR.

The `if` Statement

The if statement's basic form is

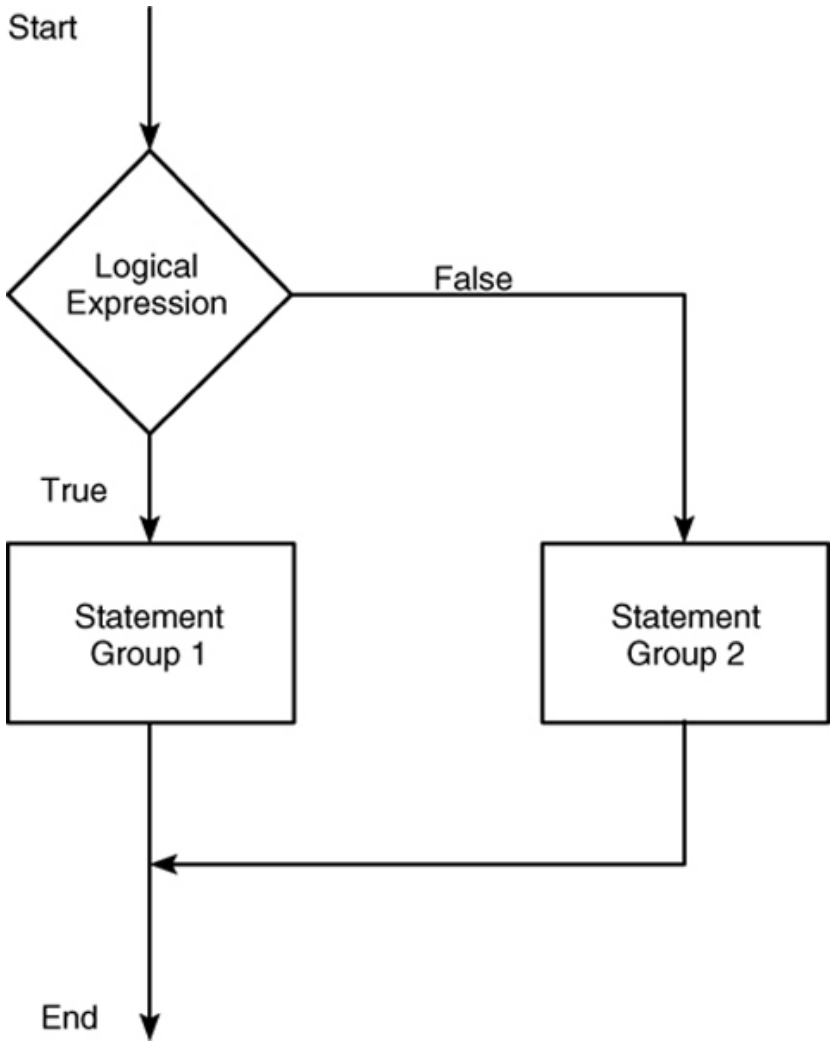
```
if logical expression
      statements
end
```



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
```



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

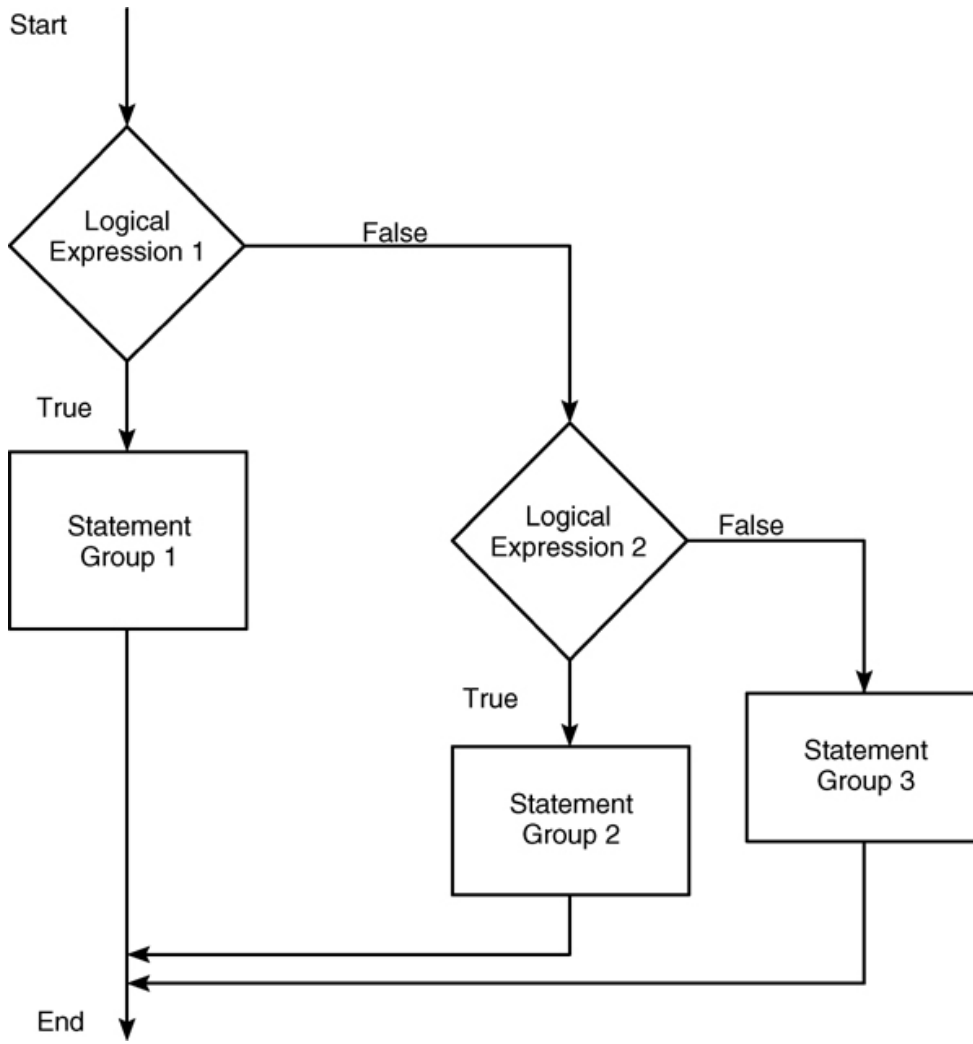
can be replaced with the more concise program

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

The `elseif` Statement

The general form of the if statement is

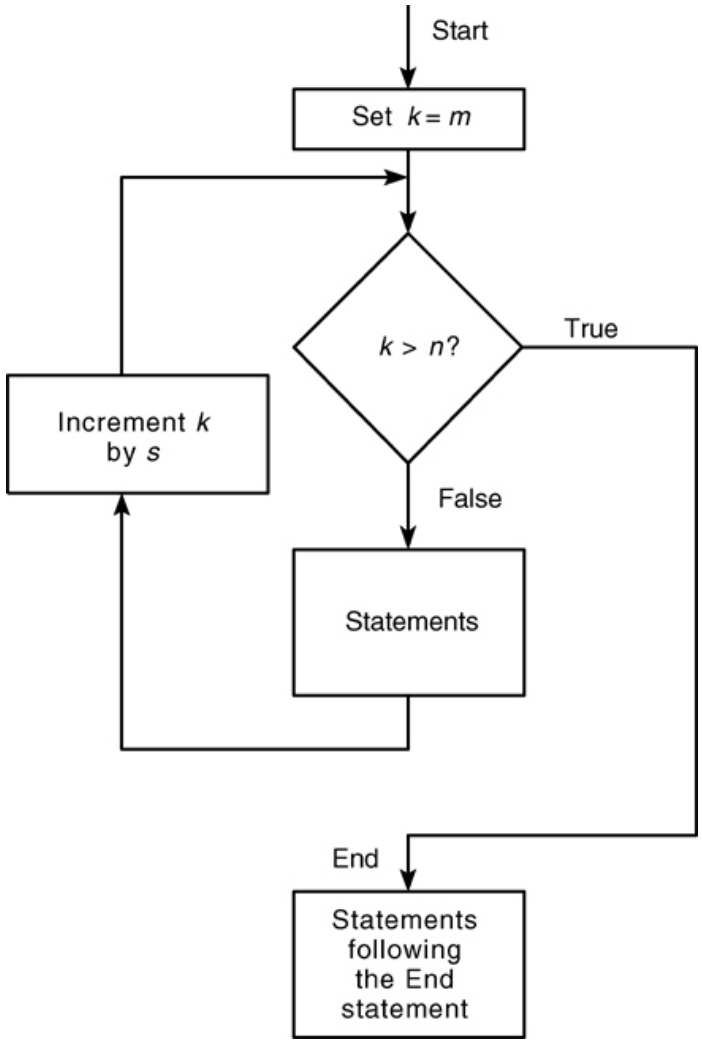
```
if logical expression 1
    statement group 1
elseif logical expression 2
    statement group 2
else
    statement group 3
end
```



for Loops

A simple example of a for loop is

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

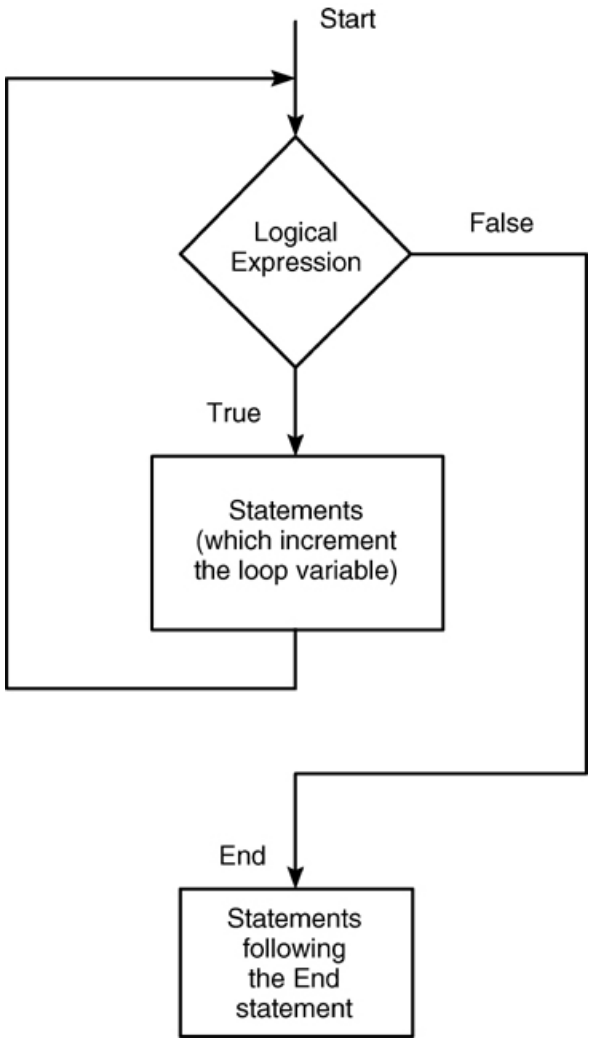


while Loops

```
while logical expression
    statements
end
```

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.

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



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
```

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

Control Flow

if

```
if w(1)==0
    % <statement>
elseif w(1)==1
    % <statement>
else
    % <statement>
end
```

switch

```
method = 'Bilinear';
switch lower(method)
   case {'linear','bilinear'}
      disp('Method is linear')
   case 'cubic'
      disp('Method is cubic')
   case 'nearest'
      disp('Method is nearest')
   otherwise
      disp('Unknown method.')
end
```

for

```
w = [];
z = 0;
is = 1:10
for i=is
    w = [w, 2*i] % Same as \/
% w(i) = 2*i
% w(end+1) = 2*i
    z = z + i;
    % break;
    % continue;
end
% avoid! same as w = 2*[1:10], z = sum([1:10]);
```

while

```
w = [];
while length(w) < 3
    w = [w, 4];
    % break
end
```

Continue and Break

The continue statement passes control to the next iteration of the for loop or while loop in which it appears, skipping any remaining statements in the body of the loop.
The break statement is used to exit early from a for loop or while loop. In nested loops, break exits from the innermost loop only.

This will never end

```
while count <= 20
    if true
        continue
    end
    count = count + 1;
end
```

This will iterate once and stop

```
while count <= 20
    if true
        break
    end
    count = count + 1;
end
```

Vectorization

MATLAB is optimized for operations involving matrices and vectors.
Vectorization: The process of revising loop-based, scalar-oriented code to use MATLAB matrix and vector operations

A simple example to create a table of logarithms:

loop-based, scalar-oriented code:

```
x = .01;
for k = 1:1001
   y(k) = log10(x);
   x = x + .01;
end
```

A vectorized version of the same code is

```
x = .01:.01:10;
y = log10(x);
```

Some functions are vectorized, hence with vectors must use element-by-element operators to combine them.
Eg: $z = e^y \sin x$, x and y vectors:

```
z=exp(y).*sin(x)
```

Vectorization

Vectorizing your code is worthwhile for:

- Appearance: Vectorized mathematical code appears more like the mathematical expressions found in textbooks, making the code easier to understand.
- Less Error Prone: Without loops, vectorized code is often shorter. Fewer lines of code mean fewer opportunities to introduce programming errors.
- Performance: Vectorized code often runs much faster than the corresponding code containing loops.

Preallocation

Another speedup techinque is preallocation. Memory allocation is slow.

```
r = zeros(32,1);
for n = 1:32
    r(n) = rank(magic(n));
end
```

Without the preallocation MATLAB would enlarge the r vector by one element each time through the loop.