Beyond the Mouse – A Short Course on Programming

4. Fundamental Programming Principles II: Control Structures (flow control)

Ronni Grapenthin

Geophysical Institute, University of Alaska Fairbanks

October 3, 2011

"The Uncomfortable Truths Well", http://xkcd.com/568 (April 13, 2009)

Intro Quiz . . .

What happens here?:

```
1 function [t lon lat height] = read_gps_data(filename)
      [t, lon, lat, height] = textread(filename, '%f%f%f%f');
3 end
```

Listing: read_gps_data.m

Intro Quiz . . .

What happens here?:

```
function [t lon lat height] = read_gps_data(filename)
    [t, lon, lat, height] = textread(filename, '%f%f%f%f');
end
```

Listing: read_gps_data.m

```
clear all, close all, clc;

gps_data = struct('time',   [], 'lon',  [], 'lat', [],...
                  'height', [], 'name', {''});

gps_data.name = 'BZ09';

[gps_data.time, gps_data.lon, gps_data.lat, ...
    gps_data.height ] = read_gps_data('BZ09.dat');

plot_gps_timeseries(gps_data);
```

Listing: plot_bz09.m

Intro Quiz . . .

What happens here?:

```
1  function plot_gps_timeseries(gps_struct)

3      figure
       subplot(3,1,1)
5      plot( gps_struct.time, gps_struct.lon-mean(gps_struct.lon) )
       title( sprintf('%s timeseries', gps_struct.name) )
7      ylabel('lon (m)');

9      subplot(3,1,2)
       plot( gps_struct.time, gps_struct.lat-mean(gps_struct.lat) )
11     ylabel('lat (m)');

13     subplot(3,1,3)
       plot( gps_struct.time, gps_struct.height-mean(gps_struct.height) )
15     ylabel('height (m)');
       xlabel('epoch');
17 end
```

Listing: plot_gps_timeseries.m

Some Comments (mostly Matlab) . . .

- You don't have to start with an empty file – that's intimidating: use old file as 'template'
- Put it in a script, unless it's a (short) one-liner (you won't use again),
- make it a habit to include '`clear all, close all, clc;`' at the beginning of your scripts
- Keep things nice and clean: definition of function in function file; use of function on command line or in script file

For Reference . . .

It's usually a good idea to check the rules for operator precedence in the documentation of a programming language.

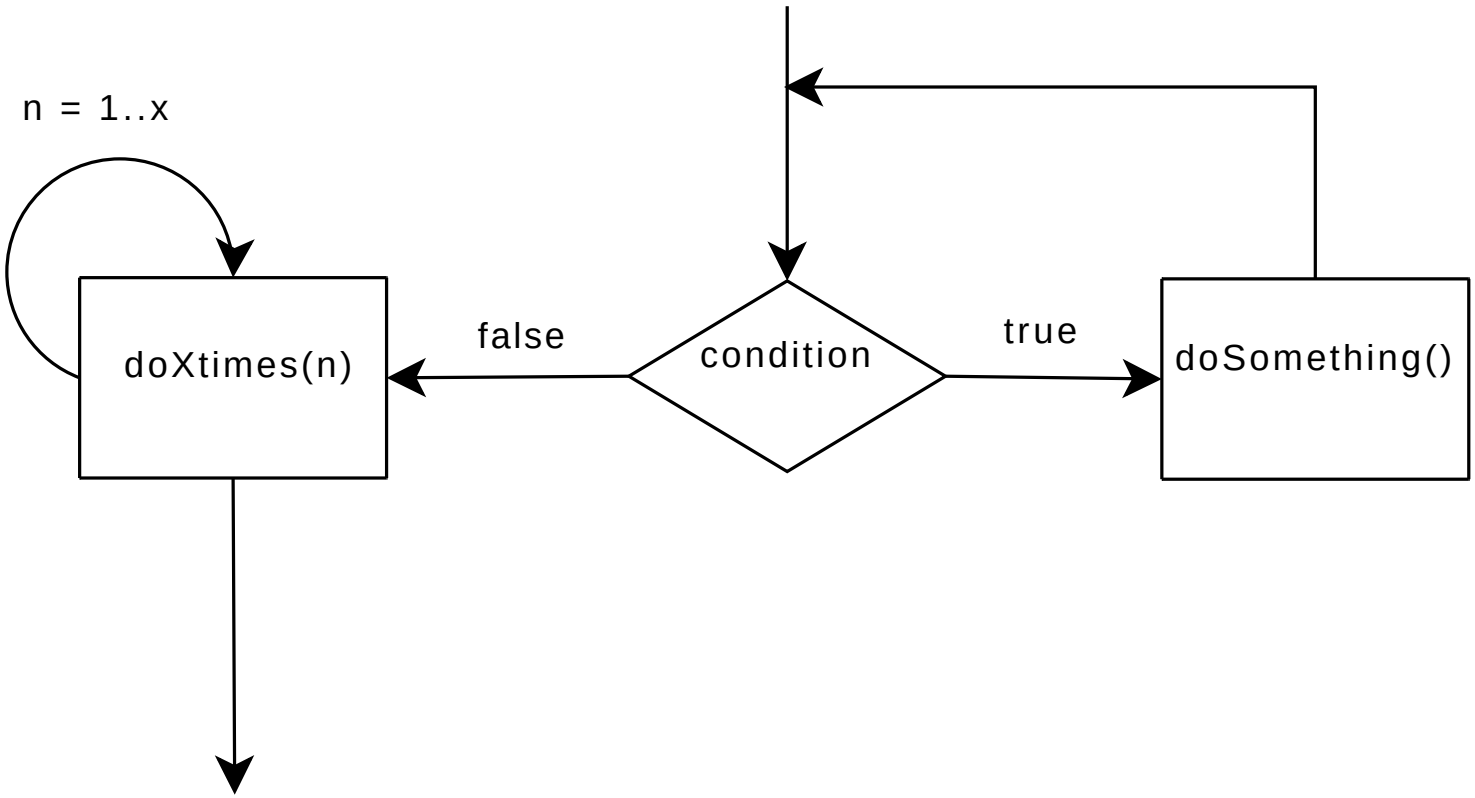
For **MATLAB** that is:

```
1.  Parentheses ()
2.  Transpose (.'), power (.^),
    complex conjugate transpose ('), matrix power (^)
3.  Unary plus (+), unary minus (-), logical negation (~)
4.  Multiplication (.*), right division (./), left division (.\),
    matrix multiplication (*), matrix right division (/), matrix left division (\)
5.  Addition (+), subtraction (-)
6.  Colon operator (:)
7.  Less than (<), less than or equal to (<=), greater than (>),
    greater than or equal to (>=), equal to (==), not equal to (~=)
8.  Element-wise AND (&)
9.  Element-wise OR (|)
10. Short-circuit AND (&&)
11. Short-circuit OR (||)
```

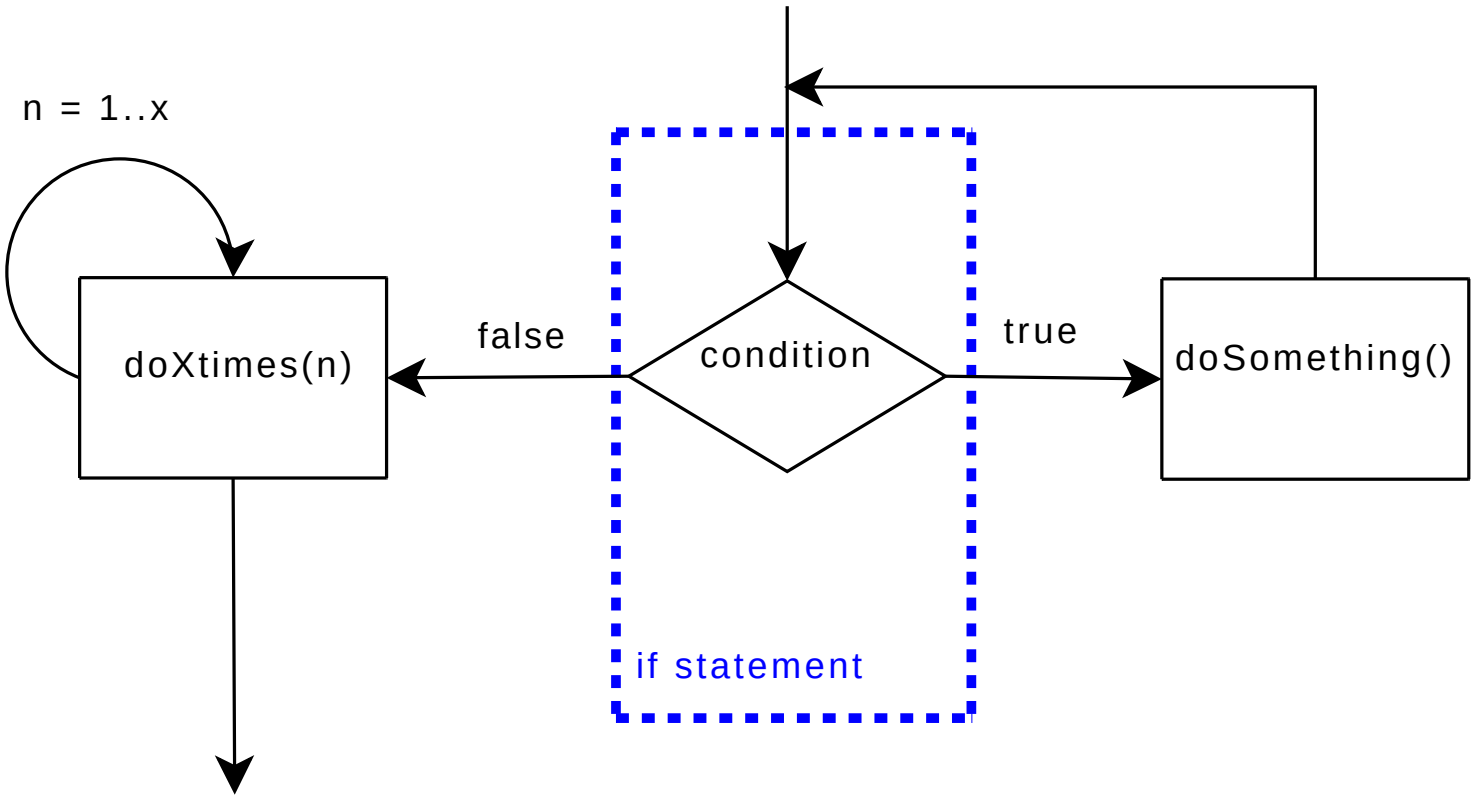
Listing: operators.txt

Keep in mind that this may be different for another programming language, read the manual!

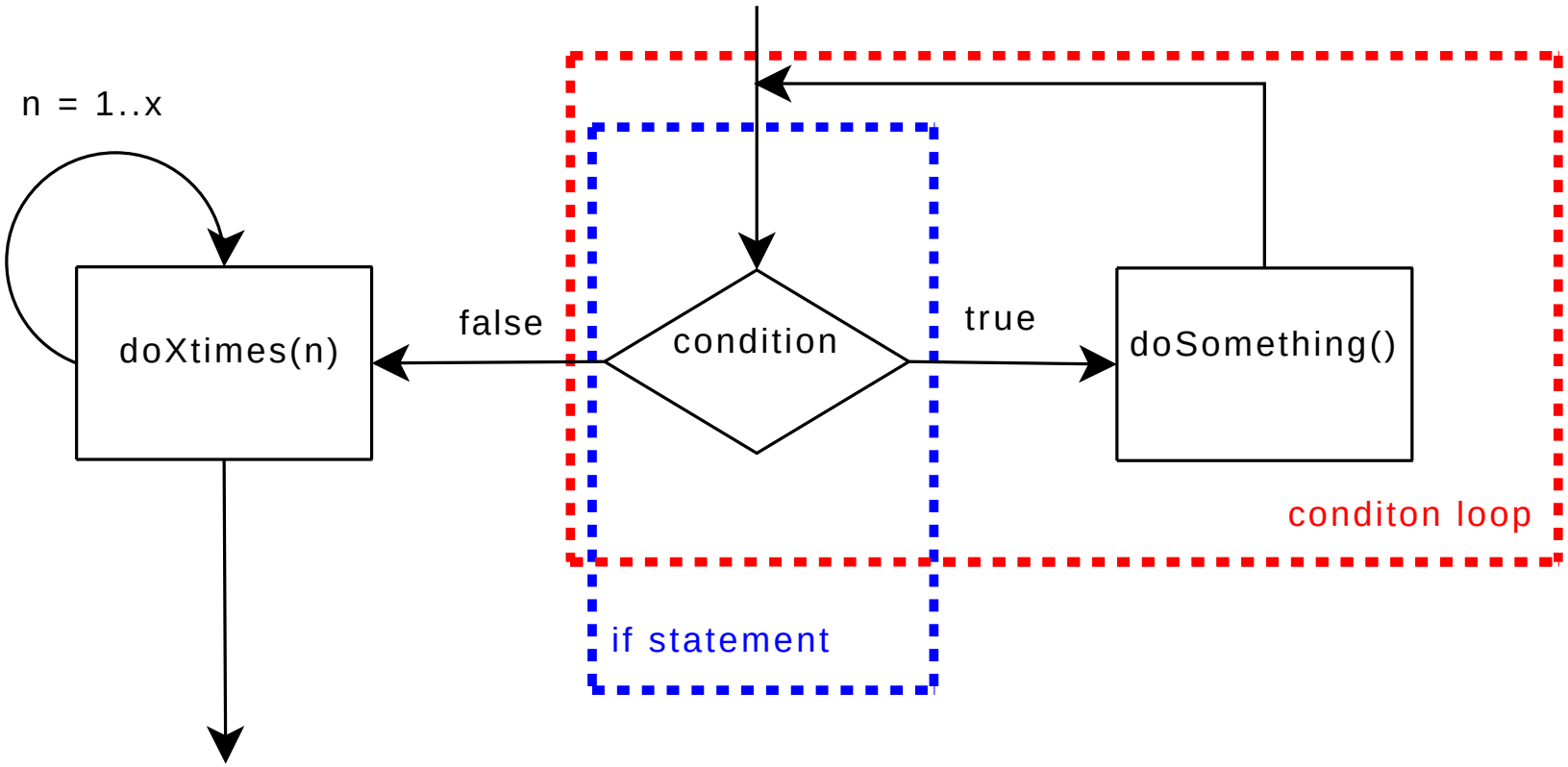
Control Flow – Redirecting the stream



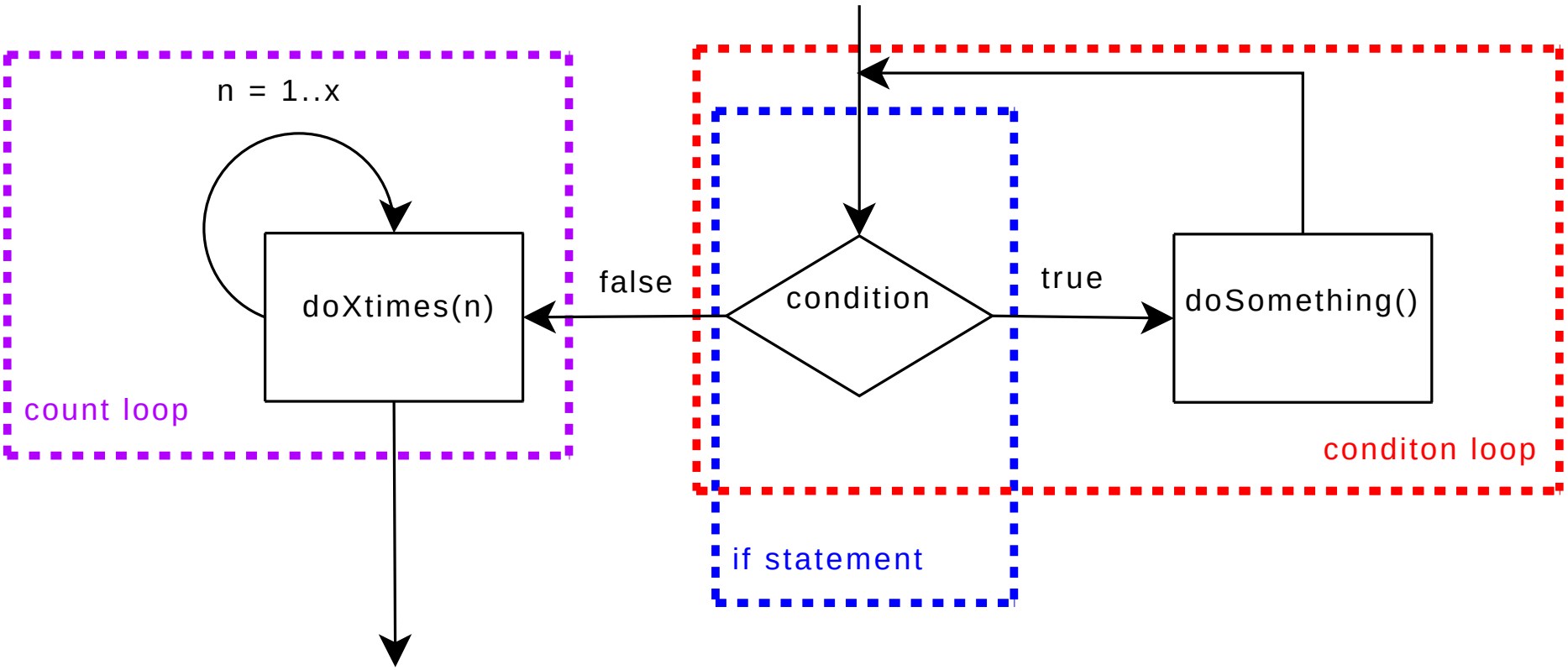
Control Flow – Redirecting the stream



Control Flow – Redirecting the stream



Control Flow – Redirecting the stream



Control Flow – Redirecting the stream

Flow control turns batch processing into programming:

- (high level) programming languages allow different behavior based on conditions **you** define – **flow control**
- A condition can be `true` (1) or `false` (0).
- You test a condition using the operators: `<, <=, >, >=, ==, != (~=)` (find equiv. in respective language)
- Functions often give numeric return values as answer to a test. In Matlab `strcmp('compare', 'strings')` will return `0` (i.e. false).

Truth Tables

Used to **compute** values of logical expressions:

Truth Tables

Used to **compute** values of logical expressions:

‘NOT’ (“~”, ’!’):

a	expression: **!a**
0	1
1	0

Truth Tables

Used to **compute** values of logical expressions:

‘NOT’ (“~”, ’!’):

a	expression: **!a**
0	1
1	0

‘AND’ (‘&&’):

a	b	expression: **a && b**
0	0	0
0	1	0
1	0	0
1	1	1

Truth Tables

Used to **compute** values of logical expressions:

‘NOT’ (“~”, ’!’):

a	expression: **!a**
0	1
1	0

‘AND’ (‘&&’):

a	b	expression: **a && b**
0	0	0
0	1	0
1	0	0
1	1	1

‘OR’ (‘||’):

a	b	expression: **a** || **b**
0	0	0
0	1	1
1	0	1
1	1	1

Truth Tables

Used to **compute** values of logical expressions:

‘NOT’ (“~”, ’!’):

a	expression: **!a**
0	1
1	0

‘AND’ (‘&&’):

a	b	expression: **a && b**
0	0	0
0	1	0
1	0	0
1	1	1

‘OR’ (‘||’):

a	b	expression: **a || b**
0	0	0
0	1	1
1	0	1
1	1	1

‘XOR’:

a	b	expression: **a xor b**
0	0	0
0	1	1
1	0	1
1	1	0

Truth Tables

Exercise for you to work through:

a	b	c	**(a && b) || c**	**(a || !b) && (a xor c)**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Control flow (0) – `statements` and such

We need a little bit of a formal definition for the following slides. Bear with me

Formal language definitions

```
1 <block> ::= { <statement list> }.

3 <statement list> ::=
          <statement>
5       | <statement list> <statement>.

7 <statement> ::=
          <block>
9       | <assignment statement>
        | <if statement>
11      | <for loop>
        | <while loop>
13      | <do statement>
        | . . .
```

Listing: bnf.txt

Control flow (0) – `statements` and such

We need a little bit of a formal definition for the following slides. Bear with me

Formal language definitions

```
1 <block> ::= { <statement list> }.

3 <statement list> ::=
        <statement>
5     | <statement list> <statement>.

7 <statement> ::=
        <block>
9     | <assignment statement>
      | <if statement>
11    | <for loop>
      | <while loop>
13    | <do statement>
      | . . .
```

Listing: bnf.txt

’[’ and ’]’ enclose optional statements

Control flow (1) – `if - then - else`

Formal

```
<if statement> ::= if (<condition>) <statement> [else <statement>].
```

Control flow (1) – `if - then - else`

Formal

```
<if statement> ::= if (<condition>) <statement> [else <statement>].
```

Matlab

```
% if ( CONDITION ) STATEMENT
% [elseif STATEMENT ]
% [else STATEMENT ]
% end.
%
% EXAMPLE: What are we gonna
% do today?
%
clc; clear all; close all;

day=weekday(now);

if (day == 6 )
   disp('PUB!')
elseif (day == 1 || day == 7)
      disp('playin''')
else
   disp('workin''')
end
```

Listing: if_example.m

Control flow (1) – `if - then - else`

Formal

```
<if statement> ::= if (<condition>) <statement> [else <statement>].
```

Matlab

```
% if ( CONDITION ) STATEMENT
% [elseif STATEMENT ]
% [else STATEMENT ]
% end.
%
% EXAMPLE: What are we gonna
% do today?
%
clc; clear all; close all;

day=weekday(now);

if (day == 6 )
   disp('PUB!')
elseif (day == 1 || day == 7)
      disp('playin''')
else
   disp('workin''')
end
```

Listing: if_example.m

C-Shell

```
#!/bin/tcsh
# if ( <condition> ) then <statement>
# [else <statement> ]
# endif
#
# Example: What are we gonna do today?

set day = `date | awk '{print $1}'`

if ($day == 'Fri' ) then
   echo 'PUB!'
else
   if ($day == 'Sat' || \
       $day == 'Sun') then
      echo "playin'"
      else
      echo "workin'"
   endif
endif
```

Listing: if_example.csh

Control flow (2) – condition controlled loop: `while`

Formal

```
<while loop> ::= while (<condition>) <block>.
```

Control flow (2) – condition controlled loop: `while`

Formal

```
<while loop> ::= while (<condition>) <block>.
```

Matlab

```
% while ( CONDITION )
%   STATEMENT
% end.
%
% EXAMPLE: Read input until user has enough
%

clc;                %clear screen

while ( ~strcmp( input('More? Y/n: ', 's'), 'n' ) )
    why
end
```

Listing: while_example.m

Control flow (2) – condition controlled loop: `while`

Formal

```
<while loop> ::= while (<condition>) <block>.
```

Matlab

```
% while ( CONDITION )
%   STATEMENT
% end.
%
% EXAMPLE: Read input until user has enough
%

clc;                %clear screen

while ( ~strcmp( input('More? Y/n: ', 's'), 'n' ) )
    why
end
```

Listing: while_example.m

C-Shell

```
#!/bin/tcsh
# while ( <condition> ) <block> end
#
# Example: Tell me my fortune

echo 'Want your   fortune? (Y/n):'

while ( $< != n)
   fortune
   echo 'More? (Y/n):'
end
```

Listing: while_example.csh

Control flow (3) – count controlled loop: `for`

Formal

```
<for loop> ::= for (<assignment>; <condition>; <assignment>) <block>.
```

Control flow (3) – count controlled loop: `for`

Formal

```
<for loop> ::= for (<assignment>; <condition>; <assignment>) <block>.
```

Matlab

```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;              %clear screen
for n=1:2:10
    fprintf(1,'n=%d\n', n);
end
disp('done.');
```

Listing: for_example.m

Control flow (3) – count controlled loop: `for`

Formal

```
<for loop> ::= for (<assignment>; <condition>; <assignment>) <block>.
```

Matlab

```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;              %clear screen
for n=1:2:10
    fprintf(1,'n=%d\n', n);
end
disp('done.');
```

Listing: for_example.m

C-Shell

```
#!/bin/tcsh
# foreach variable ( <list> ) <block>
#
# Example: list files in current
# directory.

foreach x (`ls ./`)
   echo $x
end
```

Listing: foreach_example.csh

Control flow (4) – breaking out and continuing loops: `break, continue`

Matlab

```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;              %clear screen
for n=1:10
    if(n==2)
        disp(sprintf('TWO IS PRIME!'));
        continue;
    end
    if(n==5)
        disp( ... %note the dots !!!
            sprintf('Well, that''s enough!'));
        break;
    end
    disp(sprintf('n=%d', n));
end
disp('done.');
```

Listing: for_break_example.m

Control flow (4) – breaking out and continuing loops: `break, continue`

Matlab

```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;              %clear screen
for n=1:10
    if(n==2)
        disp(sprintf('TWO IS PRIME!'));
        continue;
    end
    if(n==5)
        disp( ... %note the dots !!!
            sprintf('Well, that''s enough!'));
        break;
    end
    disp(sprintf('n=%d', n));
end
disp('done.');
```

Listing: for_break_example.m

C-Shell

```
#!/bin/tcsh
# foreach variable ( <list> ) <block>
#
# Example: list certain files in current
# directory.

clear # clear screen

foreach x (`ls ./`)
    if ($x == foreach_example.csh) then
        echo "That's me:              " $x
        continue  #<--- We continue our job
    endif

    if ($x == 'while_example.csh') then
        echo 'I could be a "while":' $x
        break #<--- We exit the foreach
    endif
end

echo "Done."
```

Listing: foreach_break_example.csh

Control flow (5) – `for` as `while`

Matlab – `for`

```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;              %clear screen
for n=1:2:10
    fprintf(1,'n=%d\n', n);
end
disp('done.');
```

Listing: for_example.m

Control flow (5) – `for` as `while`

Matlab – `for`

```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;                %clear screen
for n=1:2:10
    fprintf(1,'n=%d\n', n);
end
disp('done.');
```

Listing: for_example.m

Matlab – `while`

```
% for variable = expression
%   STATEMENT
% end.
%
% Can be translated into a while loop.
%
% EXAMPLE: count from 1 to 10
%
clc;                %clear screen

n=1;

while(n<=10)
    disp(sprintf('n=%d', n));
    n = n+2;
end
disp('done.');
```

Listing: for_as_while.m

Control flow (6) – Error control: `try-catch`

Formal

```
<try_catch> ::= try <block> catch <block>.
```

Control flow (6) – Error control: `try-catch`

Matlab

```
% try, STATEMENT, catch ME, STATEMENT, end.
% EXAMPLE: file opening
clc;
try
    fid  = fopen('whatever.txt', 'r'); % open a non-existing file
    data = fread(fid);                 % now try to get its data
    fclose(fid)
catch myException                      % define any name for an error message object
    %let the user know, implement graceful program termination ... write to stderror
    fprintf(2, '??? Error using ==> fread\n\n')     % recreate Matlab error message
    fprintf(2, '%s\n', myException.message);        % actual message from error message object
    fprintf(2, 'Error in ==> %s at %d\n\n\n', ...   % where did things occur?
                    myException.stack.name, myException.stack.line);

    fprintf(1, 'Simpler:\n')                        % use internal function to get Matlab
    fprintf(2, '%s\n', getReport(myException));     % style report
end

disp('------------> We do get here!'), pause

%now without try-catch ...
fid  = fopen('whatever.txt', 'r');
data = fread(fid);

disp('We cannot get here!')            % We'll only make it here if 'whatever.txt' exists!
```

Listing: try_catch_example.m

Don’t you ever dare to `goto`!

“GOTO”, http://xkcd.com/292

Good Practice

How to make your code readable (language independent)

- use indentations to structure your code (align comments etc)
- use meaningful variable and function names (`sec` instead of `i` and `listFiles()` instead of `lfls()`)
- decide for one formatting and naming scheme and stick to it; no matter which one it is.
- comment your code
- do not over comment your code!
- `try` and `catch` errors
- selfstudy:
  http://www.google.com/search?hl=en&q=good+programming+style&btnG=Search