Computer Programming with MATLAB

VANDERBILT UNIVERSITY

Lesson 5: Selection

by

Akos Ledeczi and Mike Fitzpatrick

Control Flow

- Sequential control
  - Sequence of commands executed one after the other
- MATLAB interpreter
  - Part of the MATLAB program that interprets and executes the various commands
  - Sequential control: default
- Control construct
  - A method by which the interpreter selects the next command to execute
  - Sequential control: default
  - Selection or Branching

if-statement

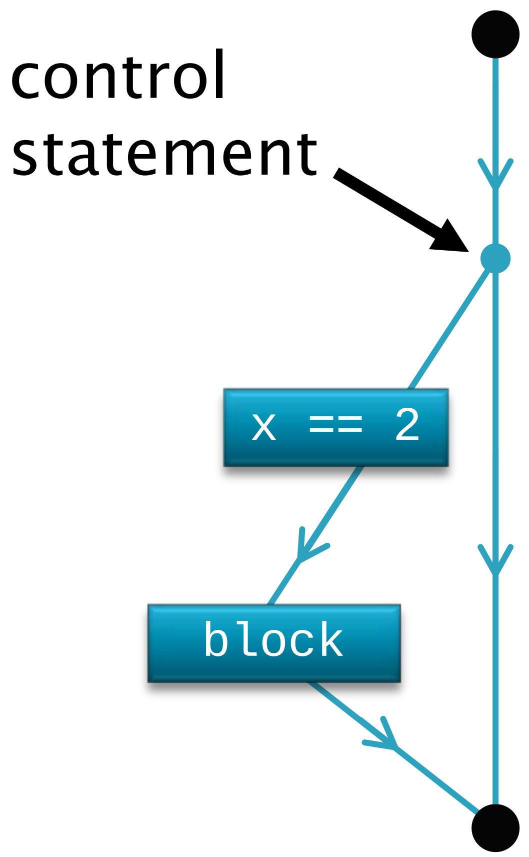
VANDERBILT UNIVERSITY

- Most common selection construct: if-statement
- Example:

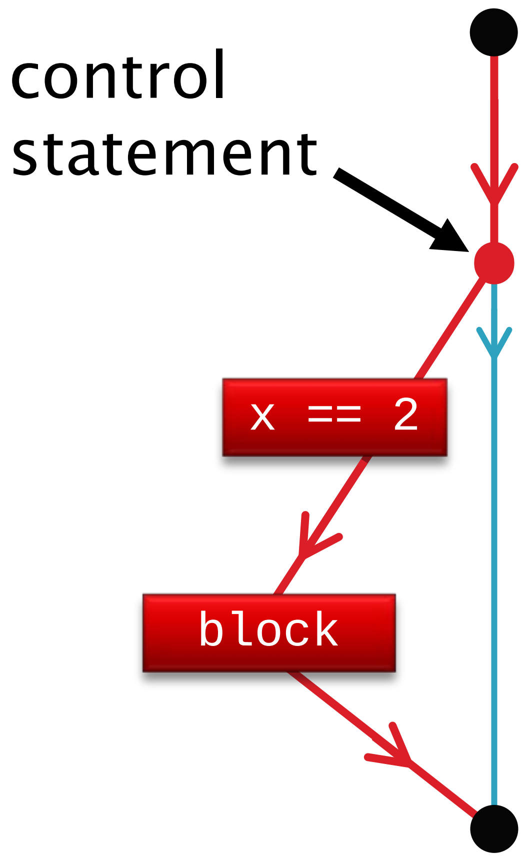
```
function guess_my_number(x)
if x == 2
  fprintf('Congrats! You guessed my number.\n');
end
```

- Begins with control statement
  - `if` keyword followed by a condition
- Ends with statement: **end**
- In between: statements to be executed if and only if condition is true

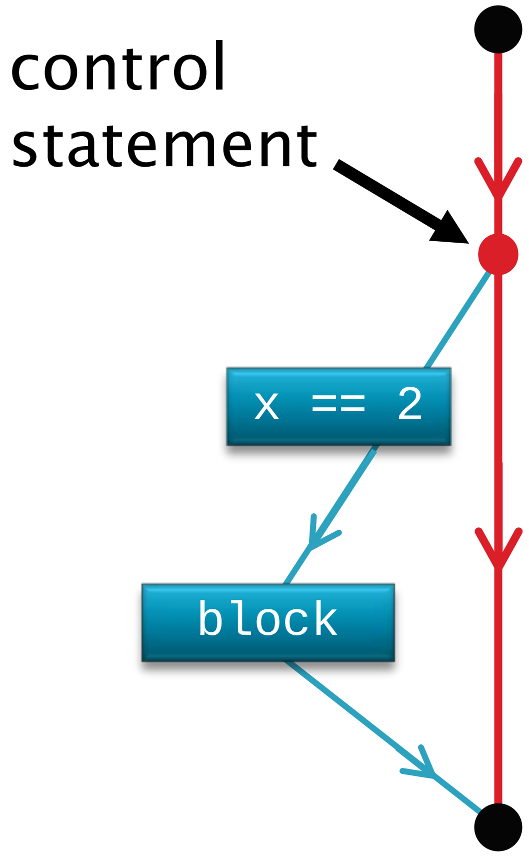
Schematic of an if-statement



Condition: true



Condition: false

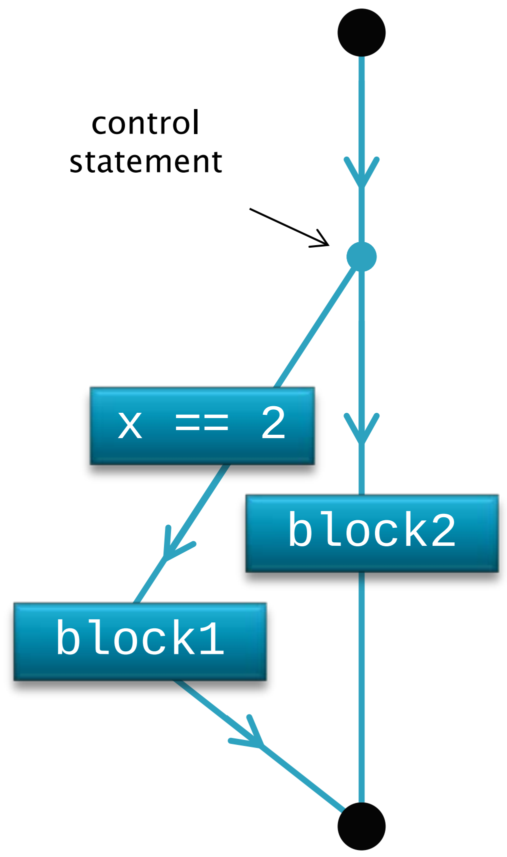


if-else-statement

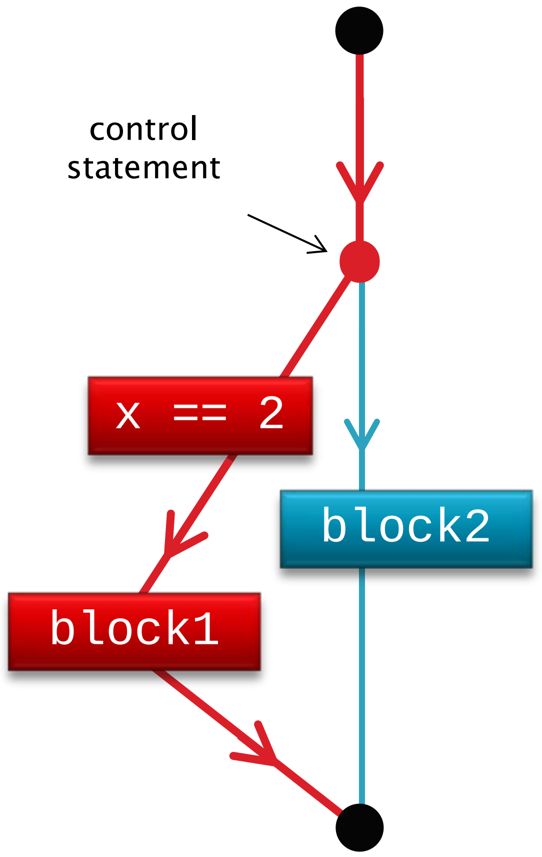
- Executing a different set of statements based on the condition:

```
function guess_my_number(x)
if x == 2
     fprintf('Congrats! You guessed my number!\n');
else
     fprintf('Not right, but a good guess.\n');
end
```

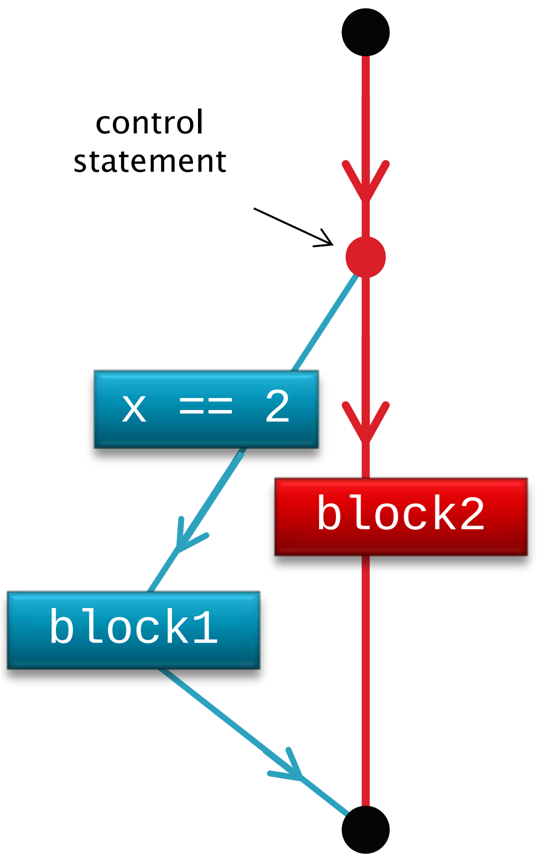
if–else–statement



Condition: true



Condition: false



if-statement summary

- if-statement:

```
if conditional
    block
end
```

Relational operators

- Produces a result that depends on the relation between its two operands
- It can appear outside if-statements!

OPERATOR	MEANING
==	is equal to
~=	is not equal to
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to

Logical operators

- Logical values:
  - Non-zero: true
  - Zero: false
  - MATLAB returns 1 for true
- How to combine logical values?
- Logical operators:

OPERATOR	MEANING
&&	and
||	or
~	not

Truth table

- not:
  - flips the value of its (single) operand
- and:
  - true if and only if both of its operands are true
- or:
  - false if and only if both of its operands are false

INPUTS		&&	| |
false	false	0	0
false	true	0	1
true	false	0	1
true	true	1	1

Truth table

- not:
  - flips the value of its (single) operand
- and:
  - true if and only if both of its operands are true
- or:
  - false if and only if both of its operands are false

INPUTS		&&	||
0	0	0	0
0	nonzero	0	1
nonzero	0	0	1
nonzero	nonzero	1	1

Precedence revisited

PRECEDENCE	OPERATOR
0	Parentheses: **(...)**
1	Exponentiation **^** and Transpose **'**
2	Unary **+**, Unary **–**, and logical negation: **~**
3	Multiplication and Division (array and matrix)
4	Addition and Subtraction
5	Colon operator **:**
6	Relational operators: **<**, **<=**, **>**, **>=**, **==**, **~=**
7	Element-wise logical “and”: &
8	Element-wise logical “or”: |
9	logical “and”: &&
10	logical “or”: ||

Precedence revisited

```
>> help precedence
         1. transpose (.'), power (.^), complex conjugate
            transpose ('), matrix power (^)
         2. unary plus (+), unary minus (-), logical negation (~)
         3. multiplication (.*), right division (./), left
            division (.\), matrix multiplication (*), matrix right
            division (/), matrix left division (\)
         4. addition (+), subtraction (-)
         5. colon operator (:)
         6. less than (<), less than or equal to (<=),
            greater than(>), greater than or equal to (>=),
            equal to (==), not equal to (~=)
         7. element-wise logical AND (&)
         8. element-wise logical OR (|)
         9. short-circuit logical AND (&&)
        10. short-circuit logical OR (||)
```

Nested if-statements

VANDERBILT UNIVERSITY

- if-statements can contain other if-statements
- Consider the example with a single if-elseif-else statement:

```
function ultimate_question(x)
if x == 42
      fprintf('Wow! You answered the question.\n');
elseif x < 42
      fprintf('Too small. Try again.\n');
else
      fprintf('Too big. Try again.\n');
end
```

Nested if-statements

- Here is a version with nesting:

```
function ultimate_question_nested(x)
if x == 42
      fprintf('Wow! You answered the question.\n');
else
      if x < 42
             fprintf('Too small. Try again.\n');
      else
             fprintf('Too big. Try again.\n');
      end
end
```

Nested if-statements

- Here is another version with nesting:

```
function ultimate_question_nested2(x)
if x <= 42
      if x == 42
             fprintf('Wow! You answered the question.\n');
      else
             fprintf('Too small. Try again.\n');
      end
else
      fprintf('Too big. Try again.\n');
end
```

Polymorphic functions

- Functions that behave differently based on
  - Number of input or output arguments
  - Type of input or output arguments
- Many built-in functions are polymorphic (sqrt, max, size, plot, etc.)
- How do we make our functions polymorphic?

Number of arguments

- Two built-in functions:
  - **`nargin`**: returns the number of actual input arguments that the function was called with
  - **`nargout`**: returns the number of output arguments that the function caller requested

Example: multiplication table

```
function [table summa] = multable(n, m)
```

- The function **`multable`** returns an n-by-m multiplication table in the output argument **`table`**
- Optionally, it can return the sum of all elements in the output argument **`summa`**
- If **`m`** is not provided, it returns and n-by-n matrix

Example

```
function [table summa] = multable(n, m)

if nargin < 2
    m = n;
end

table = (1:n)' * (1:m);

if nargout == 2
    summa = sum(table(:));
end
```

Robustness

- A function declaration specifies:
  - Name of the function,
  - Number of input arguments, and
  - Number of output arguments
- Function code and documentation specify:
  - What the function does, and
  - The type of the arguments
  - What the arguments represent
- Robustness
  - A function is robust if it handles erroneous input and output arguments, and
  - Provides a meaningful error message

Example

```
function [table summa] = multable(n, m)

if nargin < 1
    error('must have at least one input argument');
end
if nargin < 2
    m = n;
elseif ~isscalar(m) || m < 1 || m ~= fix(m)
    error('m needs to be a positive integer');
end
if ~isscalar(n) || n < 1 || n ~= fix(n)
    error('n needs to be a positive integer');
end

table = (1:n)' * (1:m);

if nargout == 2
    summa = sum(table(:));
end
```

Comments

- Extra text that is not part of the code
- MATLAB disregards it
- Anything after a % is a comment until the end of the line
- Purpose:
  - Extra information for human reader
  - Explain important or complicated parts of the program
  - Provide documentation of your functions
- Comments right after the function declaration are used by the built-in **`help`** function

Example

```
function [table summa] = multable(n, m)

%MULTABLE multiplication table.
% T = MULTABLE(N) returns an N-by-N matrix
% containing the multiplication table for
% the integers 1 through N.
% MULTABLE(N,M) returns an N-by-M matrix.
% Both input arguments must be positive
% integers.
% [T SM] = MULTABLE(...) returns the matrix
% containing the multiplication table in T
% and the sum of all its elements in SM.

if nargin < 1
    error('must have at least one input argument');
end
…
```

```
>> help multable
```

Persistent variable

- Variables:
  - Local
  - Global
  - Persistent
- Persistent variable:
  - It’s a local variable, but its value persists from one call of the function to the next.
  - Relatively rarely used
  - None of the bad side effects of global variables.