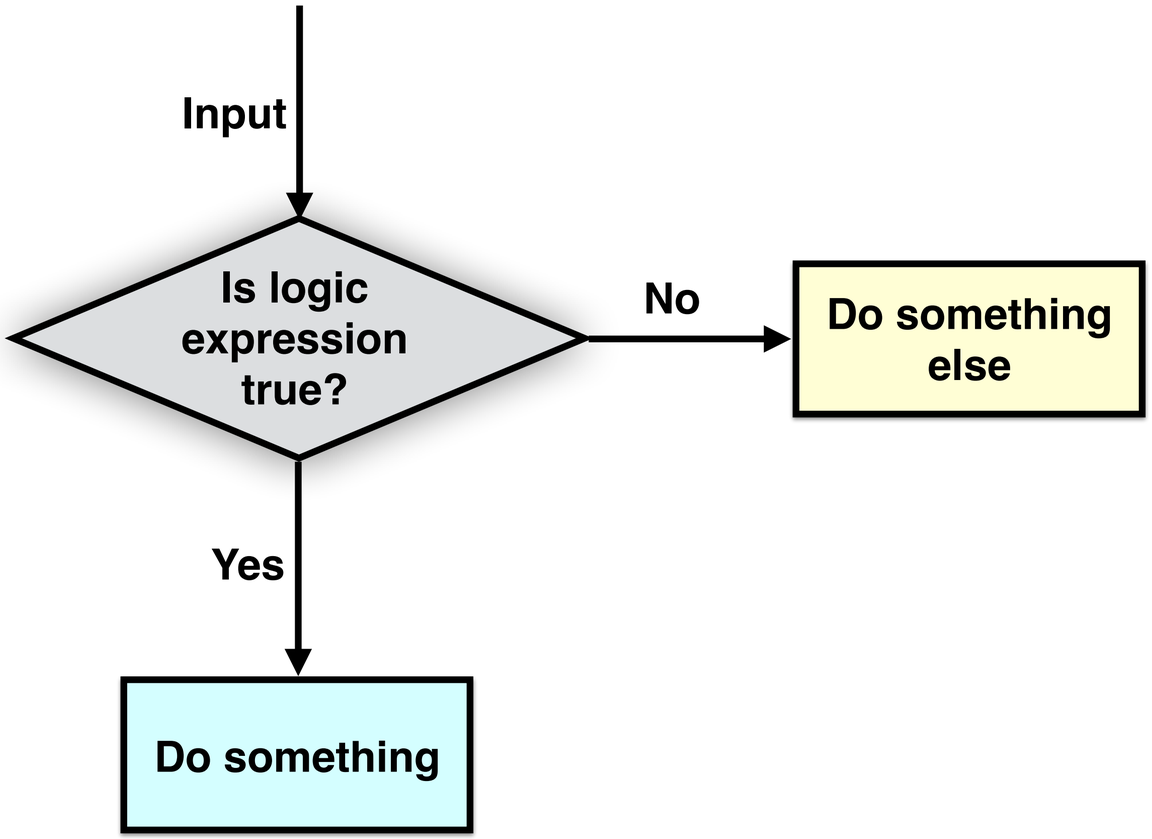
Logical Expressions and Conditionals

Concepts and Examples

CEE/MAE M20
Introduction to Computer Programming with MATLAB

At the heart of almost any computational task are ***logic and conditionals***



For such purposes, every programming language has a form of *if-else*

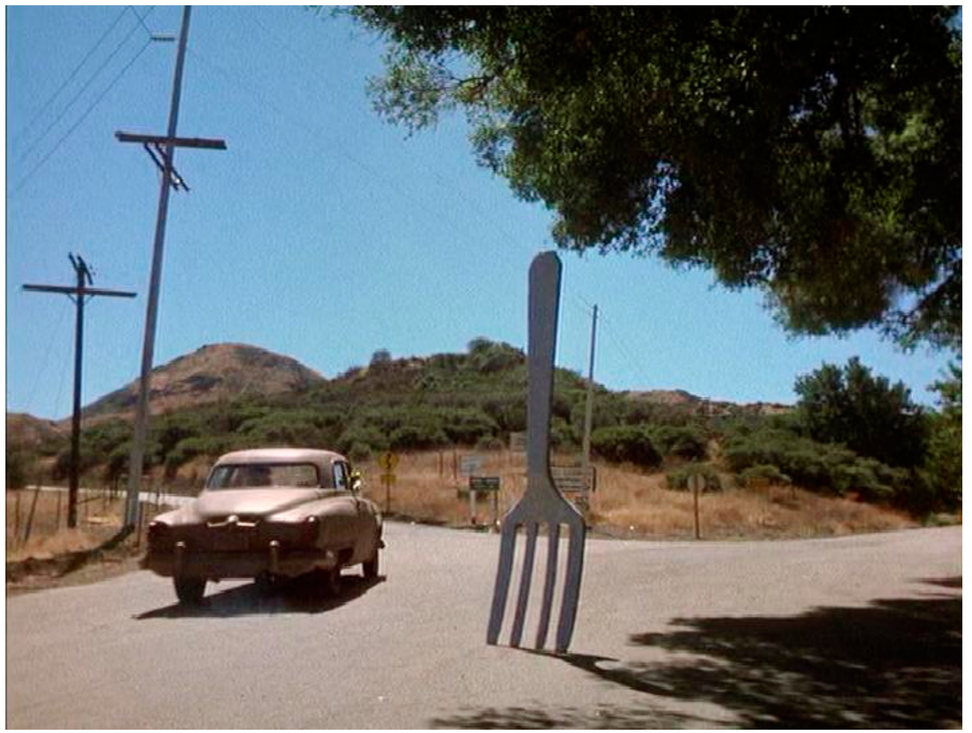
Pseudocode

```
if (expression is true)
    do something
else
    do something else
end
```

So we need to discuss
logical expressions
conditional statements

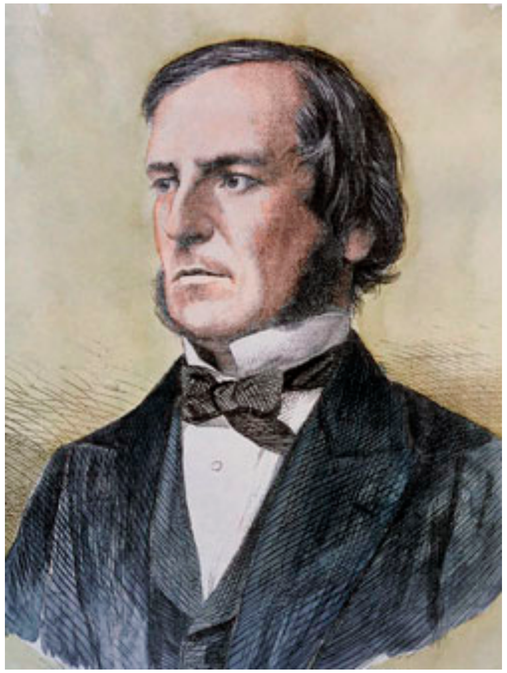
Conditionals can be more sophisticated, with more tests

```
if (logic expression 1 is true)
    do something
elseif (logic expression 2 is true)
    do something else
else
    do something completely different
end
```



George Boole (1815-1864)

... no general method for the solution of questions in the theory of probabilities can be established which does not explicitly recognise... those universal laws of thought which are the basis of all reasoning...



Logical variables = “Boolean” variables

Logical expressions

- The **logical** data type is a special type of data that can take only two values: **true** or **false**.

- In MATLAB, for example:

```
>> a = true

a =

     1

>>
```

Matlab stores this as a 1 (or 0 for false)

- How much memory does a logical variable need?
- In MATLAB, we can find this out with the **`whos`** command:

```
>> whos
  Name        Size                    Bytes  Class     Attributes

  a           1x1                         1  logical

>>
```

1 byte of memory (8 bits)

In MATLAB, all variables are stored as arrays.
A single scalar is just a 1 x 1 array.

- In contrast, MATLAB stores all numbers (integers or floating point) as double-precision floating

- For example,

```
>> b = 1;
>> whos
  Name        Size                    Bytes  Class     Attributes

  a           1x1                         1  logical
  b           1x1                         8  double

>>
```

We set this to 1 instead of ‘true’. This creates a real number, not a logical.

8 bytes of memory (64 bits)

We construct logical expressions using **relational operators** and **logic operators**

Relational operators

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

Example

```
>> 3 < 4
```

What will be the result?

Example

```
>> a = 0;
>> b = sin(pi);
>> a == b
```

What will be the result?

But sin(π)=0, so why did this happen?

```
>> b

b =

   1.2246e-16
```

Round-off error often makes values that are theoretically 0 only ‘very close’ to 0.

A simple test for equality to within machine precision…

```
>> a = 0;
>> b = sin(pi);
>> abs(a - b) < 1.0e-14;

   ans =

        1
```

Note that this issue with machine precision comes about for **all** relational operators. Beware!

Example: Is this an odd number?

The **mod** function is helpful here. Recall that **a modulo b** returns the remainder of division of a by b. In MATLAB, this is expressed as `mod(a,b)`. For example,

```
>> mod(5,3)

ans =

     2
```

← 5 divided by 3 is 1 with **a remainder of 2**

```
>> mod(16,3)

ans =

     1
```

← 16 divided by 3 is 5 with **a remainder of 1**

Example: Is this an odd number?

So to check for evenness or oddness of a integer n, use **mod(n,2)**, which will be either 0 (for even) or 1 (for odd).

For example,

```
>> n = 8;
>> is_odd = (mod(n,2)==1);
```

Is false (0) in this case, since mod(8,2) evaluates to 0

We can combine relational expressions into more complicated expressions with **logic (or Boolean) operators**

In MATLAB	Operation
`&&`	Logical AND
`||`	Logical inclusive OR
`xor`	Logical exclusive OR
`~`	Logical NOT

The results can be summarized with a **truth table**

a	b	a && b	a||b	xor(a,b)	~a
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

Example

```
>> a = 20; b = -2;
>> c = 0;  d = 1;
>> (a > b) && (c > d)
```

What will be the result?

Parentheses to ensure expressions inside are evaluated first

Example

```
>> a = 20; b = -2;
>> c = 0;  d = 1;
>> (a > b) || (c > d)
```

What will be the result?

Example: Is it a leap year?

A leap year is defined as a year that is either

- Divisible by 4 **and** not divisible by 100, **or**
- Divisible by 400

```
>> yr = 2015;
>> is_leapyear = ((mod(yr,4)==0)&&(mod(yr,100)~=0))||(mod(yr,400)==0)
```

Example: Is it a leap year?

A leap year is defined as a year that is either

- Divisible by 4 **and** not divisible by 100, **or**
- Divisible by 400

```
>> yr = 2004;
>> is_leapyear = ((mod(yr,4)==0)&&(mod(yr,100)~=0))||(mod(yr,400)==0)
```

Example: Is it a leap year?

A leap year is defined as a year that is either

- Divisible by 4 **and** not divisible by 100, **or**
- Divisible by 400

```
>> yr = 1900;
>> is_leapyear = ((mod(yr,4)==0)&&(mod(yr,100)~=0))||(mod(yr,400)==0)
```

Question: Do we need all of these parentheses?

Hierarchy of Operations

The order in which operations are evaluated:

1. All arithmetic and function expressions
2. All relational operators, from left to right
3. All ~ operators
4. All `&&` operators, from left to right
5. All `||` operators, from left to right

```
>> is_leapyear = mod(yr,4)==0 && mod(yr,100)~=0 || mod(yr,400)==0
```

Hierarchy of Operations

The order in which operations are evaluated:

1. **All arithmetic and function expressions**
2. All relational operators, from left to right
3. All ~ operators
4. All `&&` operators, from left to right
5. All `||` operators, from left to right

```
>> is_leapyear = mod(yr,4)==0 && mod(yr,100)~=0 || mod(yr,400)==0
```

Hierarchy of Operations

The order in which operations are evaluated:

1. All arithmetic and function expressions
2. **All relational operators, from left to right**
3. All ~ operators
4. All `&&` operators, from left to right
5. All `||` operators, from left to right

```
>> is_leapyear = mod(yr,4)==0 && mod(yr,100)~=0 || mod(yr,400)==0
```

Hierarchy of Operations

The order in which operations are evaluated:

1. All arithmetic and function expressions
2. All relational operators, from left to right
3. All ~ operators
4. **All `&&` operators, from left to right**
5. All `||` operators, from left to right

```
>> is_leapyear = mod(yr,4)==0 && mod(yr,100)~=0 || mod(yr,400)==0
```

Hierarchy of Operations

The order in which operations are evaluated:

1. All arithmetic and function expressions
2. All relational operators, from left to right
3. All ~ operators
4. All `&&` operators, from left to right
5. **All `||` operators, from left to right**

```
>> is_leapyear = mod(yr,4)==0 && mod(yr,100)~=0 || mod(yr,400)==0
```

Model Problem I

- Take input from the console for a start and end time. Then compute the number of hours, minutes, and seconds between the two.

Input:

Output:

```
Command Window
  Enter start hour: 4
  Enter start minute: 45
  Enter start second: 00
  Enter end hour: 8
  Enter end minute: 0
  Enter end second: 1
                    *****
  Start Time:       04:45:00
  End Time:         08:00:01
  There are 3 hours(s), 15 minutes(s), and 1 seconds(s) between the start and end times.
fx Trial>>
```

Model Problem II

- Take input from the console for a start and end date. Then compute the number of days between the two dates.

Input:

Output:

```
Command Window
  Enter start month: 12
  Enter start day: 19
  Enter start year: 1999
  Enter end month: 10
  Enter end day: 9
  Enter end year: 2014
  Total number of days between the two dates is 5408
fx Trial>>
```

Use this to check: www.timeanddate.com/date/duration.html

Some concepts reviewed

- Input/Output
- Data Types
- Arithmetic Operators
- Logical/Relational Operators
- If-Else Statements
- Switch Statement

User input

- Solicit keyboard input from the user

MATLAB Input Function

```
year = input('Enter Current Year (YYYY): ')
```

```
% check if input is valid
error('Invalid Input')
% more on this later
```

```
disp(year)
%displays the value entered for year in the
command line
```

How do we calculate expressions?

- Operators:
  - Addition +
  - Subtraction -
  - Multiplication *
  - Division /
  - Exponent ^
  - Modulo (%, mod)

```
x = 5;
% assign variable x a value of 5

y = 3*x^2 - x/2 - mod(x,3);
% evaluate expression and assign to y

disp(y);
% displays y in the command window
```

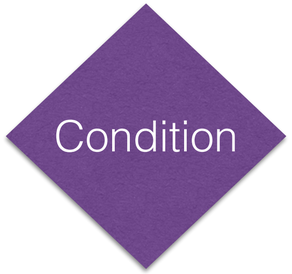
Logical/Relational Operators

- Logical Operators
  - ==, >, >=, etc.
  - Short-circuit operator
- Relational Operators
  - AND, NOT, OR, etc.

- Logical Operators:
  - AND
    `A && B, and(A,B)`
  - NOT
    `~A, not(A)`
  - OR
    `A || B, or(A,B)`
  - …

- Relational Operators:
  - Equality
    `A == B, eq(A,B)`
  - Greater than
    `A > B`
  - Greater than or equal to
    `A >= B, ge(A,B)`
  - Not equal
    `A ~= B, ne(A,B)`

If-Else Statements




```
if (month < 1 || month > 12)
    error('Invalid Data')
elseif (year < 0)
    error('Invalid Data')
…
end
```

Executing code for exclusive conditions

- A **switch** statement conditionally executes one set of statements from several choices
- Each choice is a case
- A switch block will test each case until on of the cases is true
  - When a case is true, the corresponding statements are executed and the switch block is then exited

```
% input a number and assign to x
x = input('Enter a number: ');

% switch block
switch (x)
    case -1
        disp('negative one');
    case 0
        disp('zero');
    case 1
        disp('positive one');
    otherwise
        disp('other value');
end
```

Example: The quadratic equation

Let's apply what we've learned so far on testing the roots of the quadratic equation.

$$ax^2 + bx + c = 0$$

Are the roots (1) real and distinct, (2) real and identical, or (3) complex?

The well-known roots of this equation are

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The **discriminant**, $b^2 - 4ac$, determines the behavior of the roots:

- If $b^2 - 4ac < 0$ roots are complex
- If $b^2 - 4ac = 0$ roots are real and identical
- Otherwise, roots are real and distinct

Pseudocode

```
discrim ⟵ b^2 - 4*a*c
if (discrim < 0)
    write message that equation has two complex roots
elseif (discrim == 0)
    write message that equation has two identical real roots
else
    write message that equation has two distinct real roots
end
```

The arrow means 'assign the result to variable discrim'

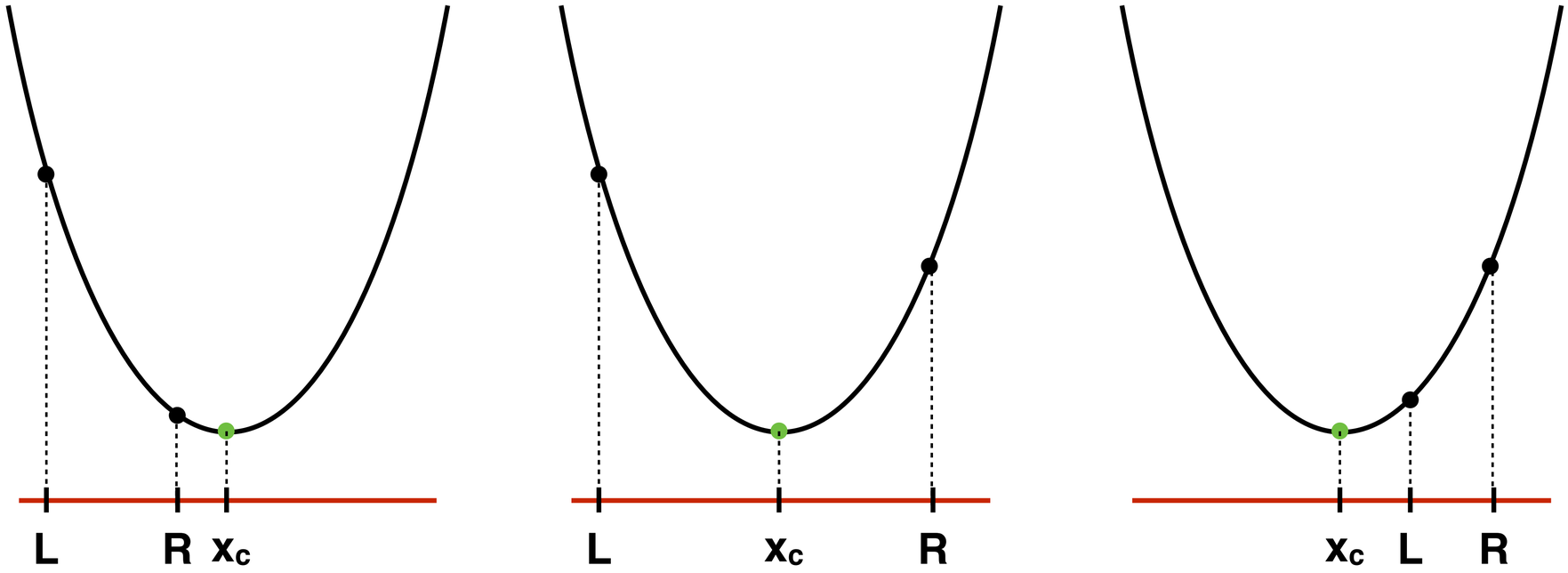
And in MATLAB code

```
discrim = b^2 - 4*a*c;
if discrim < 0
   fprintf('This equation has two complex roots.\n');
elseif discrim == 0
   fprintf('This equation has two real identical roots.\n');
else
   fprintf('This equation has two real distinct roots.\n');
end
```

Example (from ITC 1.2): Minimum of a parabola in an interval

Problem statement:
Given a quadratic $f(x) = x^2 + bx + c$, find the **minimum value** in an interval [L,R] as well as the location where the minimum occurs

Three possible scenarios



Minimum value = f(R) | **f(xc)** | **f(L)**

First, note that the overall minimum is found from

$$\frac{\mathrm{d}f}{\mathrm{d}x} = 2x + b = 0$$

The location of the minimum (the ‘critical point’) is thus

$$x_c = -b/2$$

and the value of f at the critical point is

$$f(x_c) = c - b^2/4$$

Pseudocode

```
Get inputs for b, c, L, R
xc ⟵ -b/2
if (xc < L)
      Print f(L) and L
elseif (L <= xc <= R)
      Print f(xc) and xc
else
      Print f(R) and R
end
```

```
if xc < L
   % The critical point is to the left of the interval
   fL = L^2 + b*L + c;
   fprintf('L = %6.3f f(L) = %6.3f\n',L,fL);
elseif L <= xc && xc <= R
   % The critical point is in the interval
   fxc = c - b^2/4;
   fprintf('xc = %6.3f f(xc) = %6.3f\n',xc,fxc);
else
   % The critical point is to the right of the interval
   fR = R^2 + b*R + c;
   fprintf('R = %6.3f f(R) = %6.3f\n',R,fR);
end
```

Another use of the if statement in this example is for error checking: Is the given value of R larger than the given value of L, as it should be?

```
L = input('Enter L:');
R = input('Enter R (larger than L):');
if (R < L),
   fprintf('ERROR: Invalid value for R.\n');
   return
end
```

See script **`quadmin.m`** for full code