Lecture 4: Logical operators; nesting

- Previous lecture:
  - Branching (if, elseif, else, end)
  - Relational operators (<, >=, ==, ~=, etc.)

- Today:
  - Logical operators (&&, ||, ~) and “short-circuiting”
  - More branching: nesting
  - Top-down design

- Announcements:
  - Ex2 due Sun, Feb 21
  - P1 due Tue, Feb 23
  - Partners and Academic Integrity
Problem 3

Write a code fragment that prints “Inside” if \( xc \) is in the interval and “Outside” if it is not.
Is $x_c$ in the interval $[L,R]$?

$q(x) = x^2 + bx + c$

$\bullet \ x_c = -b/2$

No!
So what is the requirement?

% Determine whether xc is in [L,R]
xc = -b/2;

if ___________________
    disp('Yes')
else
    disp('No')
end
So what is the requirement?

% Determine whether xc is in [L,R]
xc = -b/2;

if xc>=L && xc<=R
    disp('Yes')
else
    disp('No')
end
The **if** construct

```plaintext
if  boolean expression1
    statements to execute if expression1 is true

elseif  boolean expression2
    statements to execute if expression1 is false
    but expression2 is true
```

```plaintext
else
    statements to execute if all previous conditions are false
end
```
The value of a **boolean expression** is either **true** or **false**.

\[(L \leq xc) \land (xc \leq R)\]

Above (compound) boolean expression is made up of two (simple) boolean expressions. Each has a value that is either **true** or **false**.

Connect boolean expressions by **boolean operators** **and** (**\&\&**), or (**\|\|**) 

Also available is the **not** operator (**\sim**)
**Logical operators**

&&  **logical and:** Are both conditions true?  
E.g., we ask “is $L \leq x_c$ and $x_c \leq R$?”  
In our code: $L \leq x_c$ && $x_c \leq R$

||  **logical or:** Is at least one condition true?  
E.g., we can ask if $x_c$ is outside of $[L,R]$,  
i.e., “is $x_c < L$ or $R < x_c$?”  
In code: $x_c < L$ || $R < x_c$

~  **logical not:** Negation  
E.g., we can ask if $x_c$ is not outside $[L,R]$.  
In code: ~( $x_c < L$ || $R < x_c$ )
“Truth table”

X, Y represent boolean expressions.
E.g.,  \( d > 3.14 \)

| X | Y | X && Y “and” | X || Y “or” | ~Y “not” |
|---|---|-------------|-------------|----------|
| F | F | F           | F           |          |
| F | T | F           | T           | F        |
| T | F | F           | F           |          |
| T | T | T           | T           |          |
Checkpoint

- How many entries in the table are True?

A: 4
B: 5
C: 8
D: other
“Truth table”

X, Y represent boolean expressions.
E.g., \( d > 3.14 \)

| X | Y | X && Y “and” | X || Y “or” | ~Y “not” |
|---|---|-------------|-----------|--------|
| F | F | F           | F         | T      |
| F | T | F           | T         | F      |
| T | F | F           | T         | T      |
| T | T | T           | T         | F      |
“Truth table”

Matlab uses 0 to represent false, 1 to represent true

| X | Y | X && Y “and” | X || Y “or” | ~Y “not” |
|---|---|--------------|-----------|----------|
| 0 | 0 | 0            | 0         | 1        |
| 0 | 1 | 0            | 1         | 0        |
| 1 | 0 | 0            | 1         | 1        |
| 1 | 1 | 1            | 1         | 0        |
Logical operators “short-circuit”

a > b && c > d

true

Go on

Entire expression is false since the first part is false

A && expression short-circuits to false if the left operand evaluates to false.

A || expression short-circuits to _______________ if _______________

___________________
Logical operators “short-circuit”

\[ a > b \lor c > d \]

Entire expression is true since the first part is true.

A \&\& expression short-circuits to false if the left operand evaluates to \textit{false}.

A \lor expression short-circuits to true if the left operand evaluates to \textit{true}.
Why short-circuit?

- Right-hand Boolean expression may be *expensive* or potentially *invalid*
- Much clearer than alternatives

```plaintext
if (x < 0.5) || (tan(x) < 1)
    % ...
end

if (x ~= 0) && (y/x > 1e-8)
    % ...
end
```
Logical operators are required when connecting multiple Boolean expressions

Why is it wrong to use the expression $L \leq xc \leq R$ for checking if $xc$ is in $[L,R]$?

Example: Suppose $L$ is 5, $R$ is 8, and $xc$ is 10. We know that 10 is not in $[5,8]$, but the expression $L \leq xc \leq R$ gives...

$L \leq xc \land xc \leq R$
Variables $a$, $b$, and $c$ are integers between 1 and 100. Does this fragment correctly identify when lines of length $a$, $b$, and $c$ could form a right triangle?

```matlab
if a^2 + b^2 == c^2
    disp('Right tri')
else
    disp('No right tri')
end
```

A: correct  
B: false positives  
C: false negatives  
D: both B & C
This fragment prints “No” even though we have a right triangle!
a = 5;
b = 3;
c = 4;
if (a^2 + b^2 == c^2) || ...
   (a^2 + c^2 == b^2) || ...
   (b^2 + c^2 == a^2)
   disp('Right tri')
else
   disp('No right tri')
end
Consider the quadratic function

\[ q(x) = x^2 + bx + c \]

on the interval \([L, R]\):

- Is the function strictly increasing in \([L, R]\)?
- Which is smaller, \(q(L)\) or \(q(R)\)?
- What is the minimum value of \(q(x)\) in \([L, R]\)?
\[ q(x) = x^2 + bx + c \]

\[ x_c = -\frac{b}{2} \]

min at R
Conclusion

If $x_c$ is between $L$ and $R$

Then min is at $x_c$

Otherwise

Min value is at one of the endpoints
Stepwise refinement
Start with pseudocode

If $xc$ is between $L$ and $R$

Min is at $xc$

Otherwise

Min is at one of the endpoints

We have decomposed the problem into three pieces! Can choose to work with any piece next: the if-else construct/condition, min at $xc$, or min at an endpoint
Set up structure first: if-else, condition

if \( L \leq xc \text{ and } xc \leq R \)

Then min is at \( xc \)

else

Min is at one of the endpoints

end

Now refine our solution-in-progress. I’ll choose to work on the if-branch next
Refinement: filled in detail for task “min at xc”

if \ L<=xc \&\& xc<=R

% min is at \ xc

qMin= xc^2 + b*xc + c;

else

Min is at one of the endpoints

end

Continue with refining the solution... else-branch next
Refinement: detail for task “min at an endpoint”

```
if  L<=xc && xc<=R
    % min is at xc
    qMin= xc^2 + b*xc + c;
else
    % min is at one of the endpoints
    if  % xc left of bracket
        % min is at L
    else  % xc right of bracket
        % min is at R
    end
end
```

Continue with the refinement, i.e., replace comments with code
Refinement: detail for task “min at an endpoint”

\[
\text{if } \ L \leq xc \ \&\ & xc \leq R \\
\quad \text{\% min is at } xc \\
\quad qMin = xc^2 + b*xc + c; \\
\text{else} \\
\quad \text{\% min is at one of the endpoints} \\
\quad \text{if } \ xc < L \\
\quad \quad qMin = L^2 + b*L + c; \\
\quad \text{else} \\
\quad \quad qMin = R^2 + b*R + c; \\
\text{end} \\
\text{end} \]
Final solution (given $b,c,L,R,xc$)

if $L \leq xc \&\& xc \leq R$

% min is at xc
$qMin = xc^2 + b*xc + c;$

else

% min is at one of the endpoints
if $xc < L$
$qMin = L^2 + b*L + c;$
else
$qMin = R^2 + b*R + c;$
end

else

$xc < L$
$qMin = L^2 + b*L + c;$
else
$qMin = R^2 + b*R + c;$
end

See quadMin.m
quadMinGraph.m
Notice that there are 3 alternatives ➔ can use `elseif`!

```plaintext
if L<=xc && xc<=R
    % min is at xc
    qMin= xc^2 + b*xc + c;
else
    % min at one endpoint
    if xc < L
        qMin= L^2 + b*L + c;
    else
        qMin= R^2 + b*R + c;
    end
end
```

```plaintext
if L<=xc && xc<=R
    % min is at xc
    qMin= xc^2 + b*xc + c;
elseif xc < L
    qMin= L^2 + b*L + c;
else
    qMin= R^2 + b*R + c;
end
```
An algorithm is an idea. To use an algorithm you must choose a programming language and implement the algorithm.
If $xc$ is between L and R

Then min value is at $xc$

Otherwise

Min value is at one of the endpoints
if $L \leq xc \land xc \leq R$
    % min is at xc

else
    % min is at one of the endpoints

end
if \ L\leq xc \ \&\& \ xc\leq R
   \% \text{min is at } xc

else
   \% \text{min is at one of the endpoints}

end
if \quad L \leq xc \quad \&\& \quad xc \leq R

\quad \% \quad \text{min is at } xc

\quad qMin = x^c^2 + b*xc + c;

else

\quad \% \quad \text{min is at one of the endpoints}

end
if \ L<=xc \&\& \ xc<=R \\
\ \ \ \ \ \ % \ min \ is \ at \ xc \\
\ \ \ \ \ \ qMin= xc^2 + b*xc + c; \\
else \\
\ \ \ % \ min \ is \ at \ one \ of \ the \ endpoints

end
if \( L \leq xc \leq R \)
\[
\text{\% min is at } xc
\]
\[q\text{Min} = xc^2 + b*xc + c;\]
else
\[
\text{\% min is at one of the endpoints}
\]
if \( xc < L \)

else

end

end
if \ L \leq xc \ \&\ \& \ xc \leq R \\
\text{\% min is at } xc \\
qMin = xc^2 + b*xc + c;

else \\
\text{\% min is at one of the endpoints} \\
if \ xc < L \\
\text{\% min is at one of the endpoints} \\
i \\
else \\
qMin = R^2 + b*R + c;
end \\
end
Testing and debugging

- An integral part of the design loop

- The programmer’s job, not someone else’s
  - Don’t ask TAs “is this right?”; *Run your own tests*, then ask for guidance on failures

- Doesn’t need to be formal, but does need to be thought through

- Testing tips
  - Know what your immediate goal is
  - Look for simple cases, compare with hand-calcs
  - Think about corner cases – try to break things while still respecting input constraints
Checkpoint: Should we use this code to decide your grade?

```matlab
score = input('Enter score: ');
if score > 55
    disp('D')
elseif score > 65
    disp('C')
elseif score > 80
    disp('B')
elseif score > 93
    disp('A')
else
    disp('Try again')
end
```

A: yes

B: No – high scores might get low grade

C: No – low scores might get high grade

D: No – some scores might get no grade
A stick of unit length is split into two pieces. The breakpoint is randomly selected. On average, how long is the shorter piece?

Physical experiment?  
Thought experiment?  → analysis  
Computational experiment!  → simulation

Need to repeat many trials!