- Previous Lecture:
  - Branching
  - Logical operators
- Today's Lecture:
  - Logical operators and values
  - More branching—*nesting*
  - The idea of repetition
- Announcement:
  - Project 1 (P1) due today at 6pm

Logical operators

&& logical <u>and</u>: Are both conditions true?

E.g., we ask "is $L \le x_c$ and $x_c \le R$?"

In our code: `L<=xc && xc<=R`

|| logical <u>or</u>: Is at least one condition true?

E.g., we can ask if x_c is outside of $[L,R]$,

i.e., "is $x_c \le L$ or $R \le x_c$?"

In code: `xc<=L || R<=xc`

~ logical <u>not</u>: Negation

E.g., we can ask if x_c is not outside $[L,R]$.

In code: `~(xc<=L || R<=xc)`

Logical operators will short-circuit

- "It's a good thing."
- Consider the compound condition
  `L<=xc && xc<=R`
- If `L` is greater than `xc`, then the 1st condition⇒*false*. Then the entire compound condition must give *false* as well, no matter what `xc` and `R` are.
- A `&&` condition short-circuits to false if the left operand evaluates to *false*
- A `||` condition short-circuits to _________ if
  ____.

Always use logical operators to connect simple boolean expressions

Why is it wrong to use the expression

`L <= xc <= R`

for checking if x_c 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 <= xc <= R` gives...

"Truth table"

Matlab uses **0** to represent false,
1 to represent true

X	Y	X && Y "and"	X || Y "or"	~X "not"
1	1	1	1	0
1	0	0	1	
0	1	0	1	1
0	0	0	0	

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<=xc && xc<=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

Final solution (given b,c,L,R,xc)

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

An if-statement can appear within a branch—just like any other kind of statement!

January 31, 2008 Lecture 4 23

```
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
```

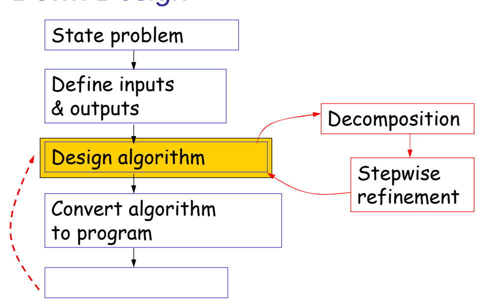
True or false: We don't need the **elseif** keyword at all (in the Matlab language).

A: true

B: false

January 31, 2008 Lecture 4 26

Top-Down Design



An algorithm is an idea. To use an algorithm you must choose a programming language and implement the algorithm.