- Previous Lecture (and Discussion):
  - Branching (`if`, `elseif`, `else`, `end`)
  - Relational operators (`<`, `>=`, `==`, `~=`, …, etc.)
  - Logical operators (`&&`, `||`, `~`)

- Today's Lecture:
  - Logical operators and "short-circuiting"
  - More branching—*nesting*
  - Top-down design

- Announcements:
  - Discussions this week in Upson B7 computer lab
  - Project 1 (P1) due Thursday at 11pm
  - Submit real .m files (plain text, not from a word processing software such as Microsoft Word)

Things to know about the `if` construct

- At most one branch of statements is executed
- There can be any number of `elseif` clauses
- There can be at most one `else` clause
- The `else` clause must be the last clause in the construct
- The `else` clause does not have a condition (boolean expression)

Consider the quadratic function

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

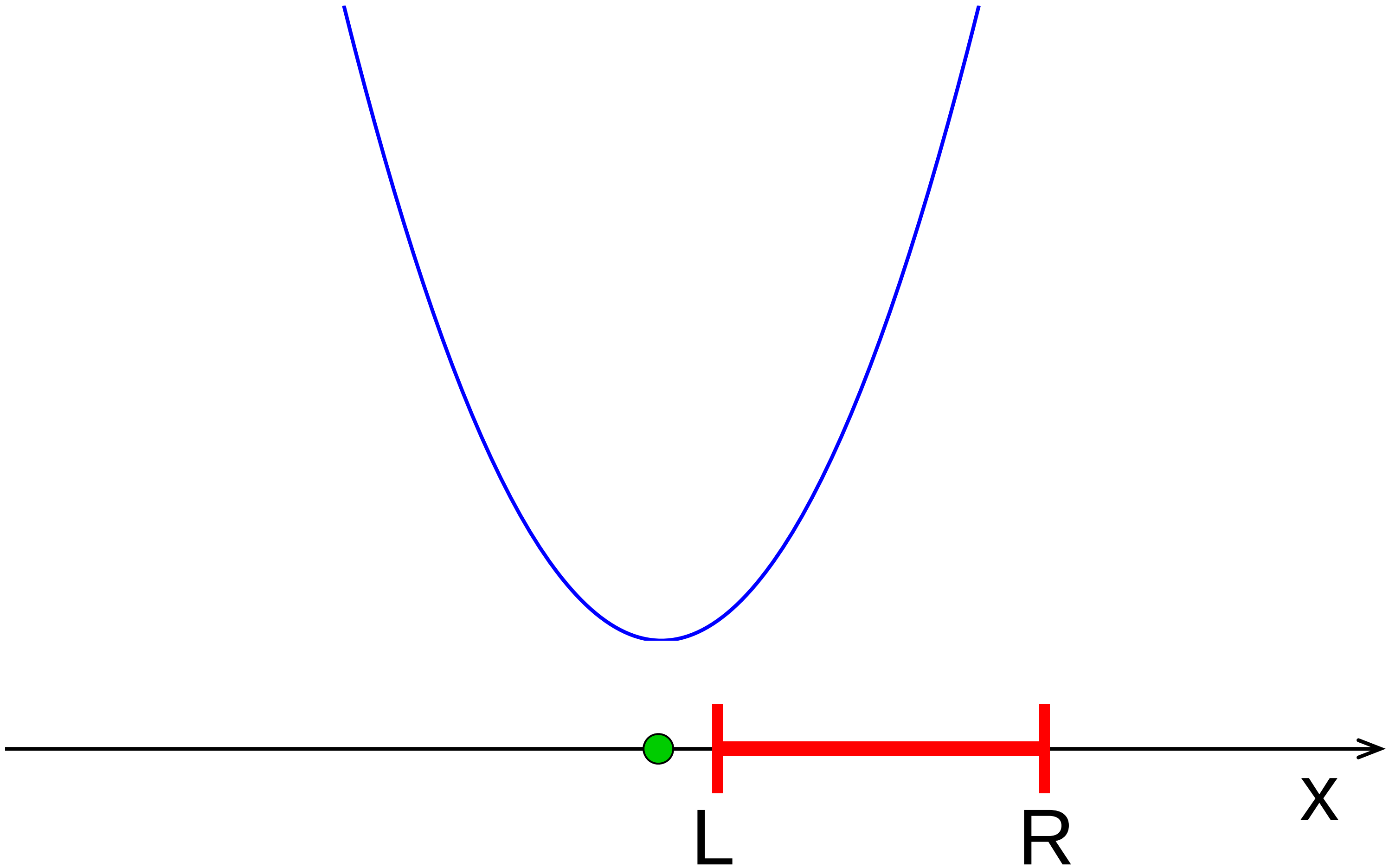
on the interval $[L , R]$:

- 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 $[L , R]$?

Minimum is at L, R, or xc

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

$$x_c = -b/2$$



Modified Problem 3

Write a code fragment that prints "yes" if xc is in the interval and "no" if it is not.

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 L<=xc && xc<=R

   disp('Yes')
else
   disp('No')
end
```

The value of a boolean expression is either true or false.

```
(L<=xc) && (xc<=R)
```

This (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	not
&&	||	~

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<=xc && xc<=R`

Logical operators

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

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

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 $x_c < L$ or $R < x_c$?"

In code: **xc<L || R<xc**

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 $x_c < L$ or $R < x_c$?"

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

&& 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 $x_c < L$ or $R < x_c$?"

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)`

“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	T			
T	F			
T	T			

“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"

$\underbrace{a > b}_{\text{true}}$ && $c > d$

Go on

$\underbrace{a > b}_{\text{false}}$ && $c > d$

Stop

Entire expression is false since the first part is false

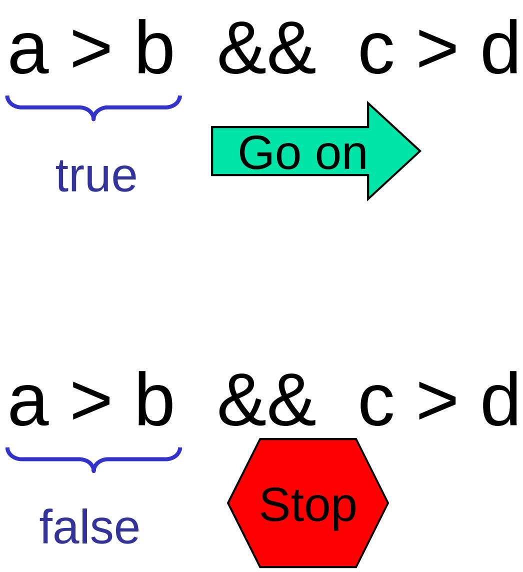
A **&&** condition short-circuits to false if the left operand evaluates to *false*.

A || condition short-circuits to ______________________ if

____

____

Logical operators “short-circuit”



Entire expression is false since the first part is false

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

A **||** condition short-circuits to true if the left operand evaluates to *true*.

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…

Variables a, b, and c have whole number values. True or false: This fragment prints "Yes" if there is a *right triangle* with side lengths a, b, and c and prints "No" otherwise.

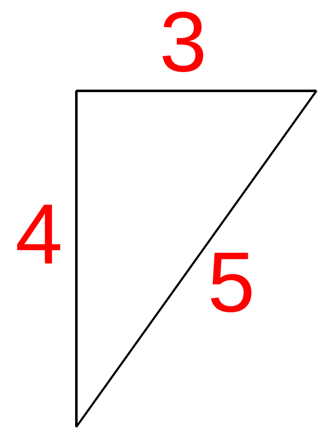
```
if a^2 + b^2 == c^2
   disp('Yes')
else
   disp('No')
end
```

A: true

B: false

```
a = 5;
b = 3;
c = 4;
if  (a^2+b^2==c^2)

    disp('Yes')
else
    disp('No')
end
```

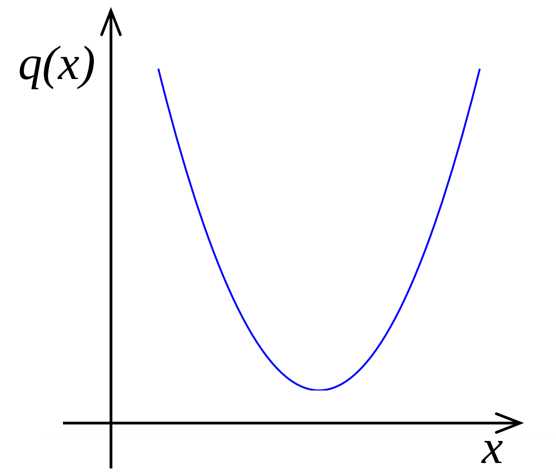


This fragment prints "No" even though we have a right triangle!

```
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('Yes')
else
   disp('No')
end
```

Consider the quadratic function

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

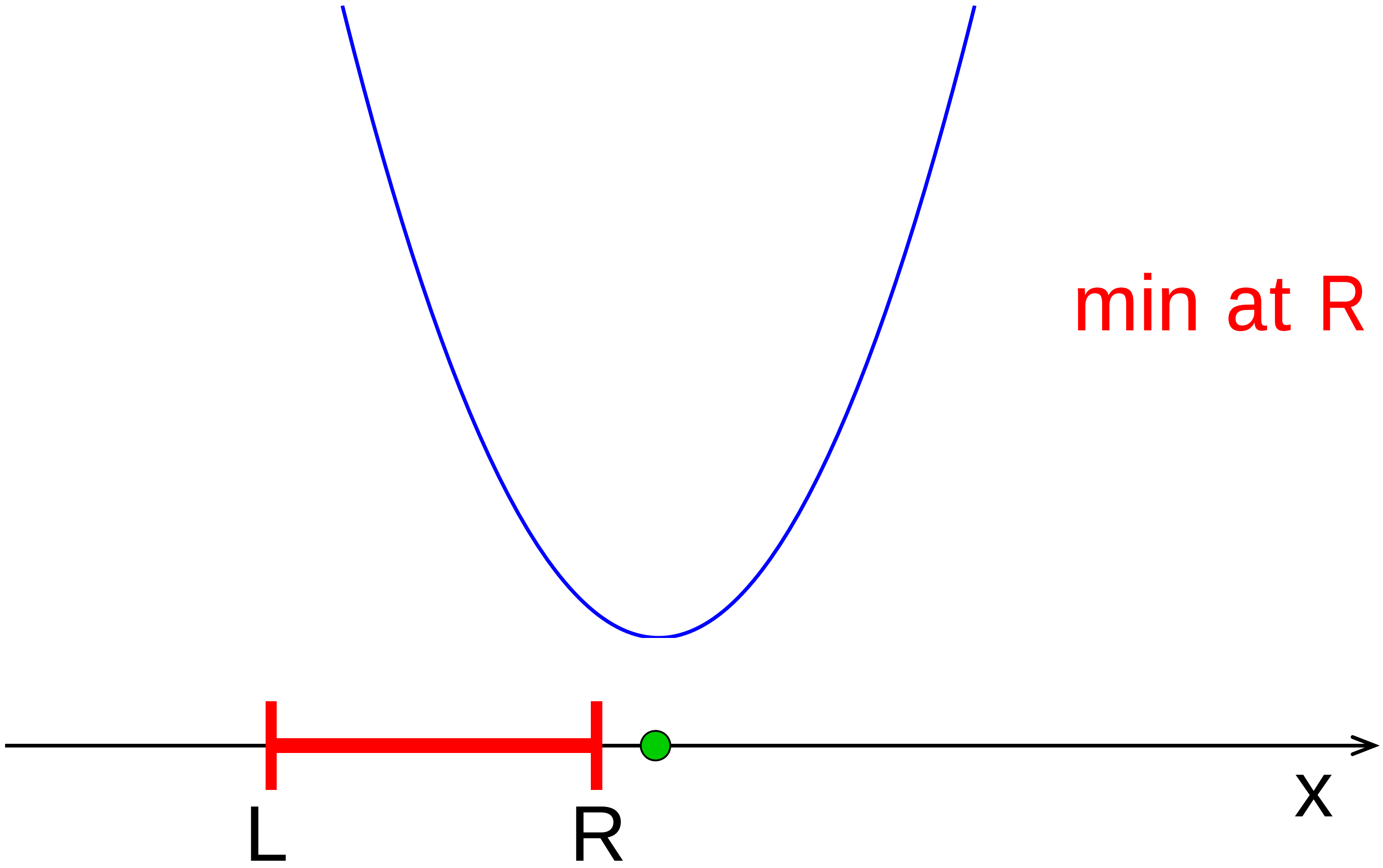


on the interval $[L , R]$:

- 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 $[L , R]$?

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

$x_c = -b/2$



Conclusion

If x_c is between L and R

Then min is at x_c

Otherwise

Min value is at one of the endpoints

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

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

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!

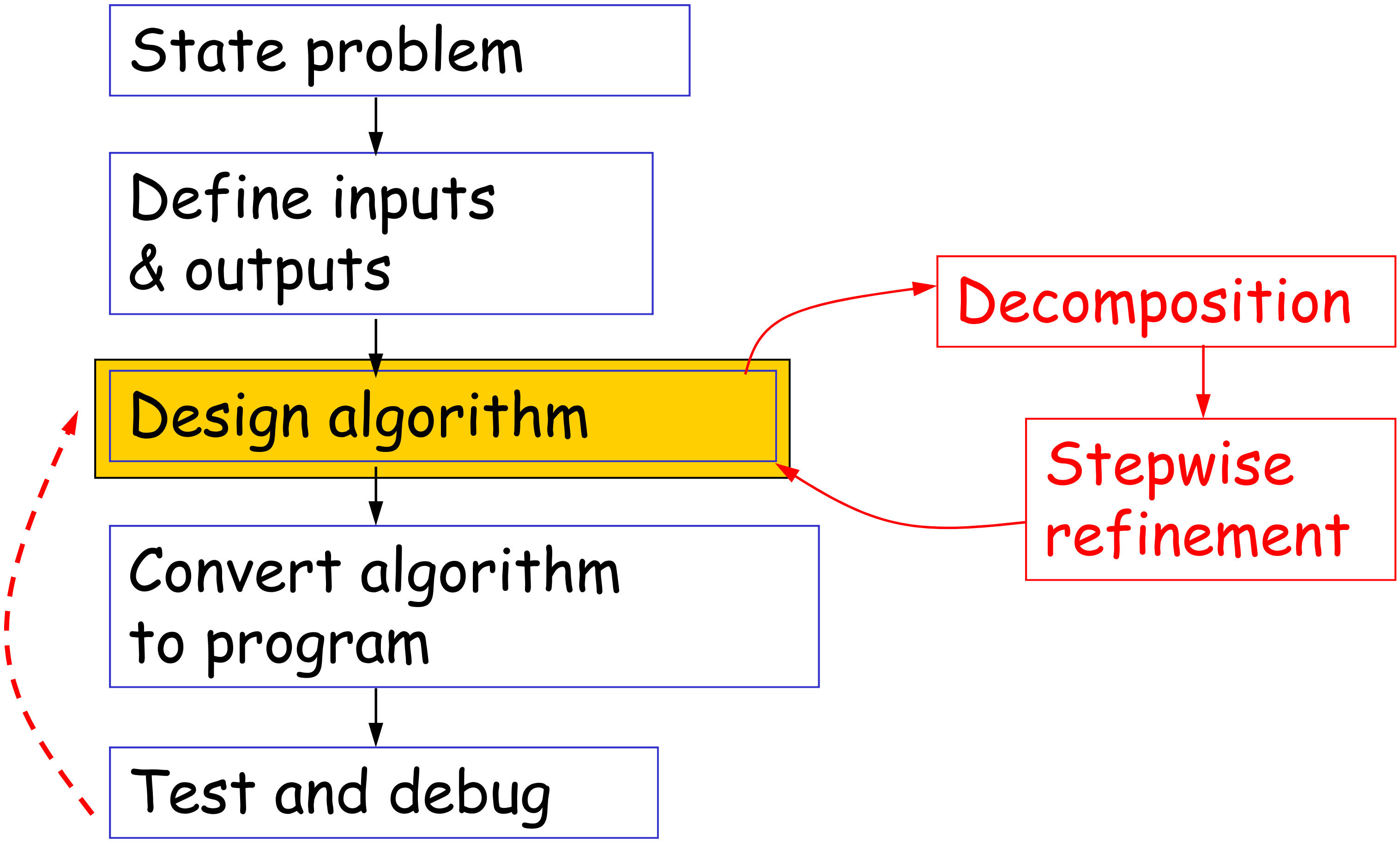
quadMin.m
quadMinGraph.m

Notice that there are 3 alternatives→can use elseif!

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

```
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
```

Top-Down Design



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