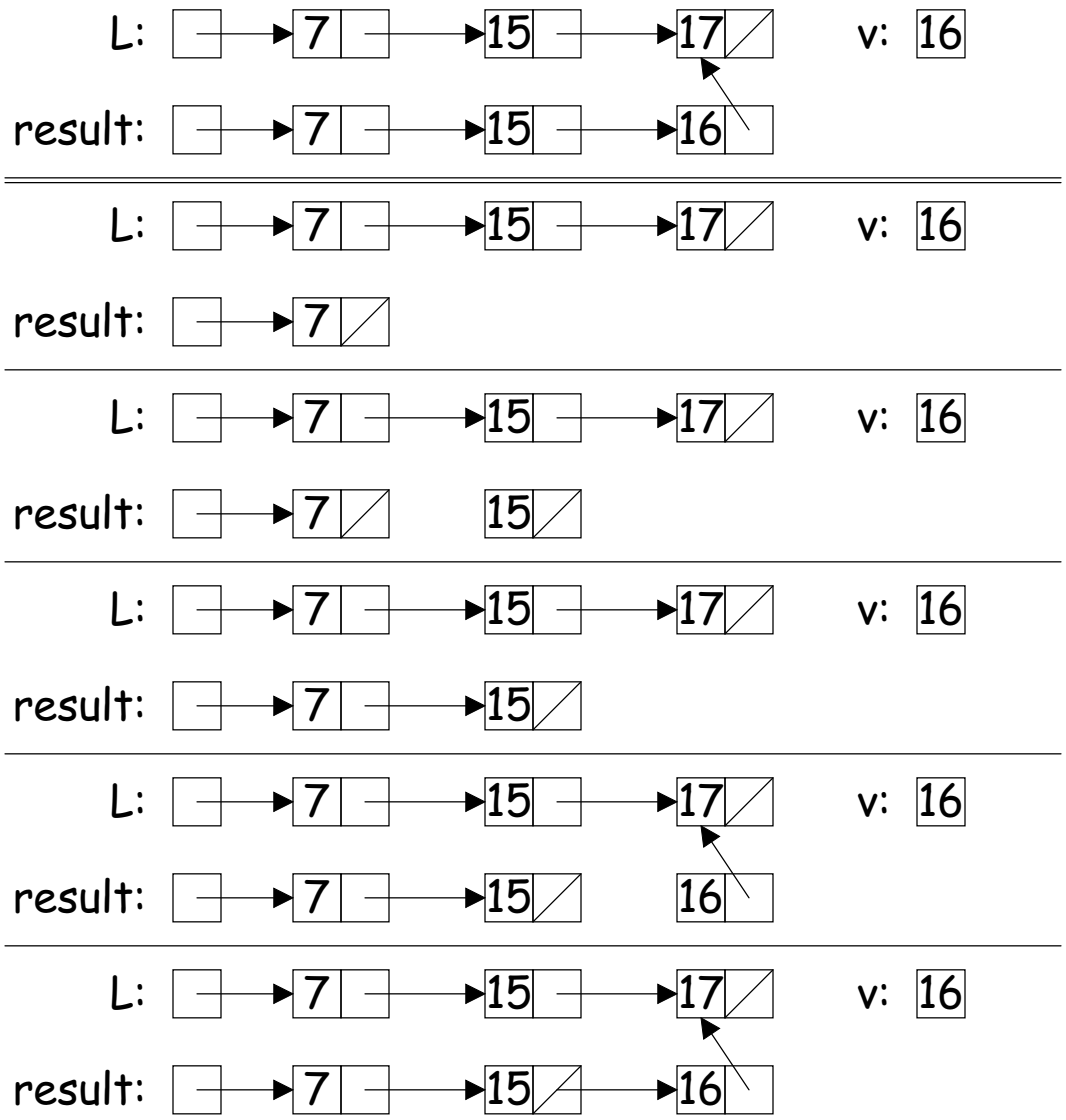
Lecture #22: Assorted Examples

Announcements

- Homework #5 due date moved to Monday after spring break (3/28) at midnight.

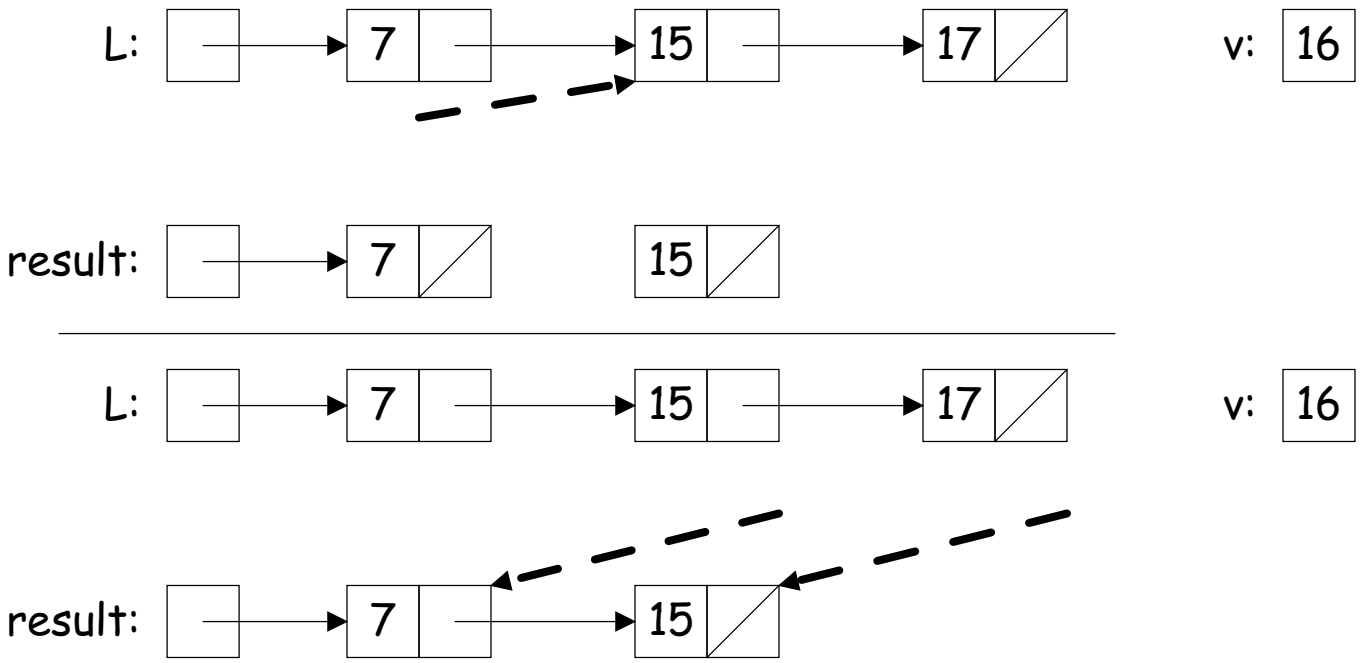
Approching a Linked-List Problem



Problem: Insert `v` into ordered list `L` nondestructively.

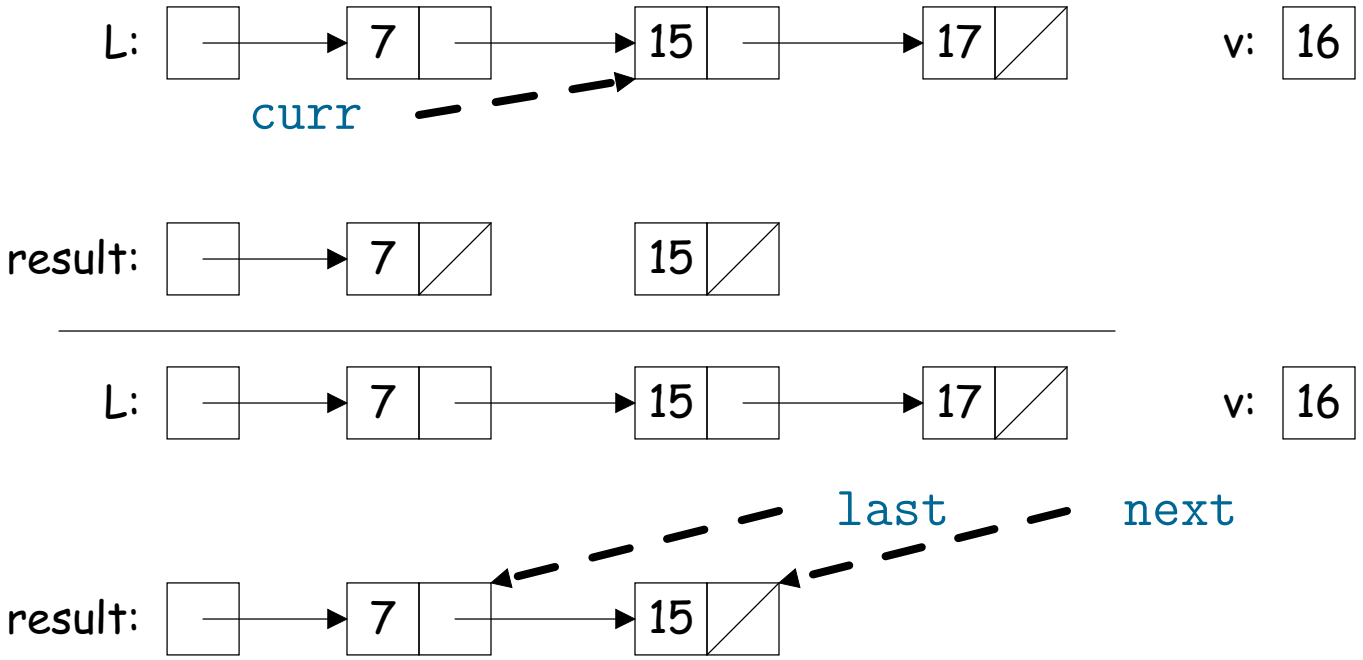
Breaking It Down

- Here are the repeated steps:



- Dashed arrows show information needed to do each step.
- These are the variables you'll need.
- After using them, must move them into position for next time.

Loop Code




```
next = Link(curr.first)
last.rest = next
# For next loop:
curr = curr.rest
last = next
```

A Worked Example: Decoding with Trees

- In Unicode or ASCII, characters are represented in interally with *fixed-length* groups of bits:

Character	Bits
'a'	0000000001100001
'0'	0000000000110000
':'	0000000000111010
'π'	0000001111000000
'♡'	0010011001100001

- These are easy to deal with internally; characters represented as pairs of 8-bit bytes.
- Obvious where one ends and the next begins.
- But not very compact.

Prefix Codes

- An alternative is to use a *variable-length code* in which characters are represented with varying numbers of bits.
- Shorter sequences of bits can represent more common characters, minimizing total length of message.
- But the input is just a sequence of bits; have to decide where each character starts in this sequence.
- One approach is to use a *prefix code*:

  The bit sequence representing a character is never a prefix of the bit sequence representing any other character.

- So, can always tell when a character ends: one more bit will yield an invalid sequence. No special separator needed.

Example

Character	Bits	Character	Bits
A	1110	N	1010
B	101100	O	1101
C	01000	P	101101
D	11111	Q	001011010
E	011	R	1000
F	00110	S	1001
G	111100	T	000
H	0101	U	01001
I	1100	V	001010
J	001011001	W	111101
K	0010111	X	001011011
L	10111	Y	00100
M	00111	Z	001011000

So, "CAT" encodes as

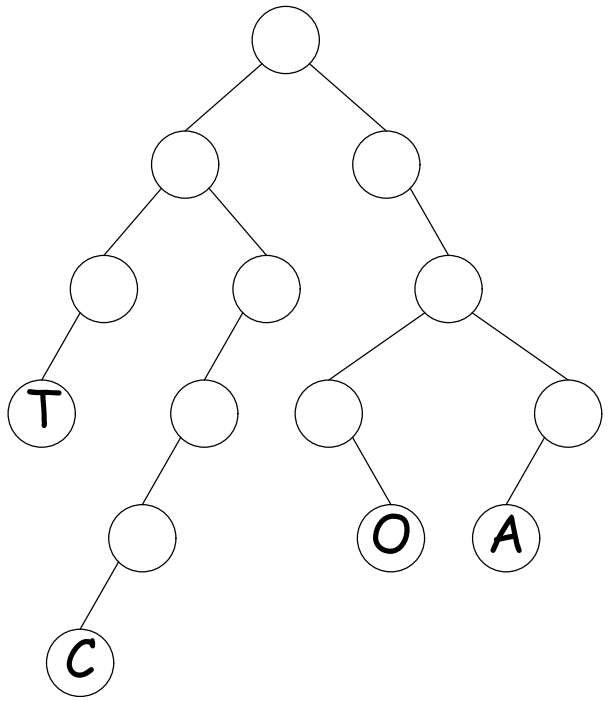
```
000000000100001100000000010000010000000001010100
C                   A                   T
```

using plain Unicode, but using the code above:

```
010001110000
C    A   T
```

Problem: How to decode?

- Can't just look up a character code until we know where it ends.
- Our plan is to consume bits (actually, we'll use strings of characters '0' and '1' for simplicity) from the left.
- Emit a decoded character whenever we determine that we've accumulated a complete character code.
- A binary tree is one way to do this.




```
A: 1110
C: 01000
O: 1101
T: 000
```

For each character:

- Start at top.
- Go left on each '0'.
- Go right on each '1'.
- Stop at a node labeled with a decoded character.

Coding and Decoding

See 22.py file for Python3 source code for performing coding, decoding, and tree-building for this data structure.