Code Generation for Pointers
l-values vs r-values

• Remember the distinction between **l-values** and **r-values**:
  • L-value: an address that can be loaded from or stored to
  • R-value: a piece of data that can be computed with

• Up until now, the only l-values we have had are variables (global variables, local variables)
l-values, r-values, and pointers, oh my!

• Semantically, what do & and * do?
• Convert between l-values and r-values!
• **Address-of operator**: take an l-value (an address) and treat it as an r-value (a piece of data)

```c
& x + 1
```

take the *address* of x, treat it as a piece of data, and add 4 to it

```c
x + 1
```

take the *value* of x, then add 1 to it
l-values, r-values, and pointers, oh my!

• Semantically, what do & and * do?
• Convert between l-values and r-values!
• De-reference operator: take an r-value (a piece of data) and treat it as an l-value (an address)

\[(x + 1)\]

take the value in x, add 4 to it, then treat the result as an address so you can load from it or store to it

\[x + 1\]

take the value of x, then add 1 to it

• Note that if the expression passed to * is an l-value, you load from it first to get an r-value, just like before
next: pointer codegen example