Arrays
what are arrays?

• An array is just a series of boxes stored consecutively in memory
  • In some languages arrays are objects (store length, etc.)
  • In C/C++, arrays are just regions of memory
what are arrays?

• An array is just a series of boxes stored consecutively in memory
  • In some languages arrays are objects (store length, etc.)
  • In C/C++, arrays are just regions of memory

\[\text{int } x[5]\]
what are arrays?

• An array is just a series of boxes stored consecutively in memory
  • In some languages arrays are objects (store length, etc.)
  • In C/C++, arrays are just regions of memory

```c
int x[5]
```

• So how do we deal with arrays?
syntactic sugar

• C/C++ arrays are an example of **syntactic sugar**
  • New language constructs that are just alternate (simpler?) ways of expressing an existing mechanism in the language
  • Example:
    
    ```
    (* ptr).x === ptr -> x
    ```

• Some languages, like Python, are full of syntactic sugar

    ```
    a[i] === a.__getitem__(i) === type(a).__getitem__(a,i)
    ```
**desugaring**

- Syntactic sugar does not require introducing new mechanisms to the language.

- It requires *translating* syntactic sugar back to its other representation: **desugaring**.

- Code generation: desugar the construct, then generate code for the *underlying construct*. 

---
Arrays in different languages

- Arrays in C/C++ are as simple as they get:
  - Just a sequence of boxes in memory

- Arrays in other settings are more complicated
  - More information: e.g., Java arrays, or STL vectors track size
  - More functionality: e.g., STL vectors can grow if more elements are added to the data structure
  - More safety: e.g., check to make sure that array accesses are not out of bounds

- Means that arrays are not just pure syntactic sugar. May involve other machinery
but

• Array-like data structures in other languages are often *backed* by C-like arrays
  • The underlying data is stored in consecutive boxes in memory

• In that case, the generated code to access the array still winds up looking like:
  1. Get base pointer address
  2. Add offset to base pointer
  3. Dereference computed address

• Fundamentally, C-like arrays are always accessed the same way (C’s implementation of arrays just happens to be a close match to the low-level representation)
next: desugaring arrays