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

int x
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]
```
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

  ```
  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:
  \[(\ast \text{ ptr}).x == \text{ ptr} \rightarrow x\]

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

  \[a[i] == a.__getitem__(i) == \text{type(a).__getitem__(a, i)}\]
• 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 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)
Desugaring Arrays
what is the syntactic sugar?

• 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

\[
\text{int } x[5]
\]

• So how do we deal with arrays?

• Arrays are essentially pointers with special syntax!
allocating arrays

• An array is a **base pointer** plus a size
  
  • Base pointer is *just a pointer* that points to the beginning of the array
  
  • Size defines number of boxes in array

• Allocating an array is just assigning a pointer:

```c
int * p
p = malloc(10 * 4) //allocate an array of 10 integers
```
allocating arrays

• An array is a **base pointer** plus a size
  
  • Base pointer is *just a pointer* that points to the beginning of the array
  
  • Size defines number of boxes in array

• You may see explicit array syntax for global/stack allocation:

  ```
  int p[10]; //allocates 10–integer array on stack
  ```

• In this case, p is still just an **int** *pointer* with some extra compiler smarts
  
  ```
  (p == &p)
  ```
Accessing arrays is very simple syntactic sugar:

\[ a[\text{expr}] \equiv \star (a + 4 \star \text{expr}) \]

size of data type pointed to by a
code generation for arrays

• Can generate code by implementing a desugaring pass
  • Before code generation, walk over AST, replace array nodes with corresponding pointer-based expression
  • Can generate code by implementing desugaring during code generation
composition of sugaring

• Desugaring composes!

\[
a[i][j] === \star (a[i] + 4 \star j) === \star (\star (a + 4 \star i) + 4 \star j)
\]
are arrays just pointers?

• Syntactic sugar can be complicated

• In some sense, yes! Array accesses are exactly equivalent to pointer arithmetic + a dereference, and pointers that point to a dynamically allocated array work as above

• But in another sense, no. If arrays are declared as arrays, with either local or global allocation, they have an array type and C/C++ do some magic with them:

  ```
  int a[4] vs int * b = malloc(16)
  ```

• a refers to the whole box, a returns \&a

• b is a pointer that points to a separate array: