```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

fun map (f,xs) =
    case xs of
        [] => []
      | x::xs' => (f x)::(map(f,xs'))

val a = map (increment, [4,8,12,16])
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman

Optional: Closure Idioms Without Closures in C

Now C

- Closures and OOP objects can have "parts" that do not show up in their types

- In C, a *function pointer* is only a code pointer
  - So without extra thought, functions taking function-pointer arguments will not be as useful as functions taking closures

- A common technique:
  - Always define function pointers and higher-order functions to take an extra, explicit environment argument
  - But without generics, no good choice for type of list elements or the environment
    - Use **`void*`** and various type casts…

The C trick

Don't do this:

```
list_t* map(void* (*f)(void*), list_t xs){
   … f(xs->head) …
}
```

Do this to support clients that need private data:

```
list_t* map(void* (*f)(void*,void*)
            void* env, list_t xs)   {
   …  f(env,xs->head) …
}
```

List libraries like this are not common in C, *but callbacks are!*

- *Always define callback interfaces to pass an extra* **`void*`**
- Lack of generics means lots of type casts in clients ☹