```
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
2013

Weak Typing

Why weak typing (C/C++)

Weak typing: There exist programs that, by definition, *must* pass static checking but then when run can "set the computer on fire"?

- Dynamic checking is optional and in practice not done
- Why might anything happen?

- Ease of language implementation: Checks left to the programmer
- Performance: Dynamic checks take time
- Lower level: Compiler does not insert information like array sizes, so it cannot do the checks

Weak typing is a poor name: Really about doing *neither* static nor dynamic checks

- A big problem is array bounds, which most PLs check dynamically

What weak typing has caused

- Old now-much-rarer saying: “strong types for weak minds”
  - Idea was humans will always be smarter than a type system (cf. undecidability), so need to let them say “trust me”

- Reality: humans are really bad at avoiding bugs
  - We need all the help we can get!
  - And type systems have gotten much more expressive (fewer false positives)

- 1 bug in a 30-million line operating system written in C can make an entire computer vulnerable
  - An important bug like this was probably announced this week (because there is one almost every week)

Example: Racket

- Racket is ***not*** weakly typed
  - It just checks most things dynamically*
  - Dynamic checking is the *definition* – if the *implementation* can analyze the code to ensure some checks are not needed, then it can *optimize them away*

- Not having ML or Java's rules can be convenient
  - Cons cells can build anything
  - Anything except `#f` is true
  - …

  This is nothing like the "catch-fire semantics" of weak typing

*Checks macro usage and undefined-variables in modules statically

Another misconception

What operations are primitives defined on and when an error?

- Example: Is `"foo" + "bar"` allowed?
- Example: Is `"foo" + 3` allowed?
- Example: Is `arr[10]` allowed if `arr` has only 5 elements?
- Example: Can you call a function with too few or too many arguments?

This is not static vs. dynamic checking (sometimes confused with it)

- It is "what is the run-time semantics of the primitive"
- It is related because it also involves trade-offs between catching bugs sooner versus maybe being more convenient

Racket generally less lenient on these things than, e.g., Ruby