What's 51 about?

Programming isn't hard.

Programming **well** is **very** hard.

We want you to write code that is:

- Reliable, efficient, readable, testable, provable, maintainable… **elegant**!

Expand your problem-solving skills:

- Recognize problems & map them onto the right languages, abstractions, & algorithms.

Course Focus

"Software Engineering in the Small"

- Introduce new programming abstractions
  - e.g., closures, abstract & algebraic data types, polymorphism, modules, classes & inheritance, synchronization, patterns, etc.
  - increase your computational tool-box, stretch your thinking.
- Introduce engineering design
  - e.g., coding style, interface design, efficiency concerns, testing.
  - models & analytic tools (e.g., big-O, evaluation models.)
  - learn to analyze, think, and express with precision.

Who should take this course?

- CS concentrators & minors should:
  - knowledge & experience is crucial for upper-level, software-intensive courses (compilers, OS, networking, AI, graphics, etc.)
  - 51 : build *up* abstractions ; 61: drive *through* abstractions
- Also electrical engineering, statistics, [applied] math, systems & synthetic biology, finance, economics, etc.
  - these fields (and many others) demand computational thinking.
- Entrepreneurs
  - engineering take on design is invaluable.
- Necessary background:
  - basic programming, algorithms, data structures (CS50)
  - mathematical "sophistication" (calc, ideally algebra)

Course Tools

We'll be using two very different programming environments.

- get used to learning languages (not that hard once you've absorbed representatives from major genres.)

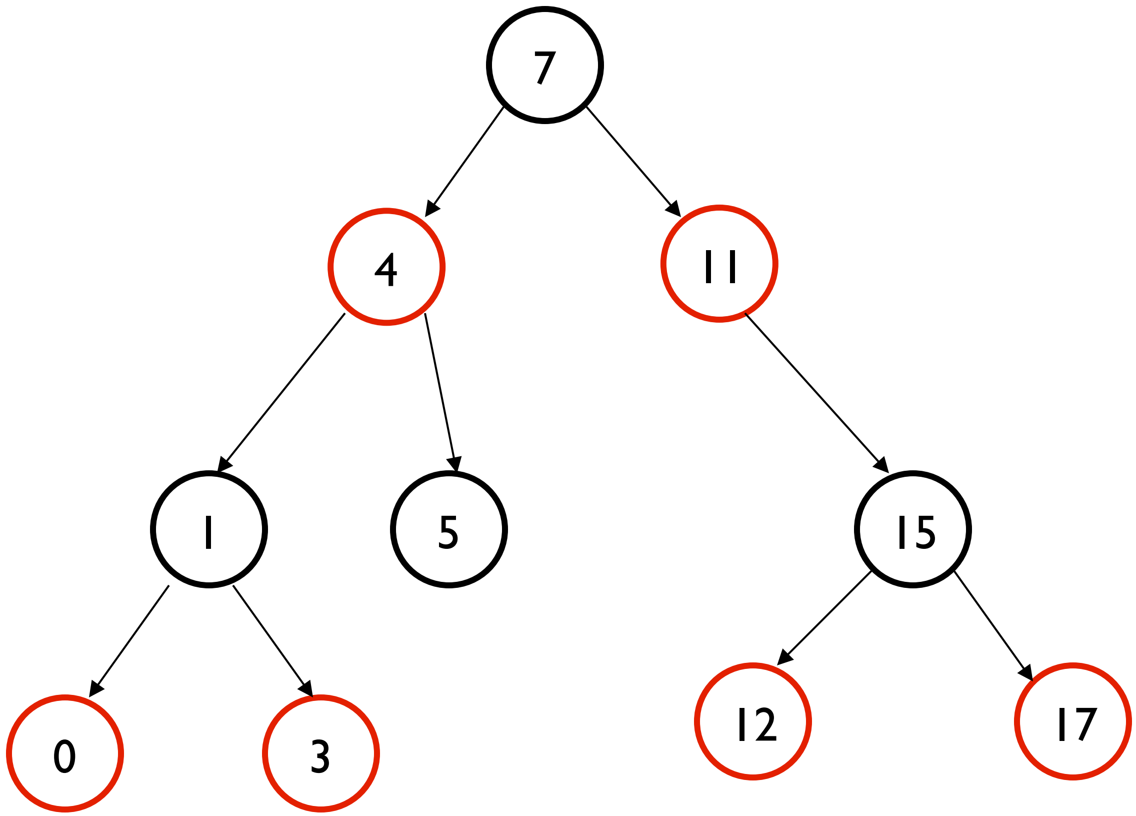
- Objective Caml (a.k.a. Ocaml & F#): First 2/3^{rds} of the class
  - functional & higher-order programming
  - functional patterns
  - substitution & environment models of evaluation
  - types, polymorphism
  - abstract data types, interfaces, modules
- Java: Final 1/3rd of the class
  - imperative & object-oriented programming
  - encapsulation, classes, subtyping, inheritance
  - concurrency, synchronization, message passing
  - OO design patterns

Language & Code

- Language & abstractions matter.
  - Try formulating an algorithm to multiply Roman numerals.
- Often, don't have the luxury of choosing the language.
  - We can still conceptualize & prototype using the right language abstractions.
  - If we understand relationships between linguistic abstractions, we can realize the code in any language.

Example: Red-Black Trees

- A particular kind of balanced search tree [Guibas & Sedgewick 1978].



C code (part 1/4)

```
void rb_insert( Tree T, node x ) {
  tree_insert( T, x );
  x->colour = red;
  while ( (x != T->root) && (x->parent->colour == red) ) {
      if ( x->parent == x->parent->parent->left ) {
         y = x->parent->parent->right;
        if ( y->colour == red ) {
         x->parent->colour = black;
          y->colour = black;
          x->parent->parent->colour = red;
        x = x->parent->parent;
      } else {
       if ( x == x->parent->right ) {
          x = x->parent;
          left_rotate( T, x );
       }
       x->parent->colour = black;
       x->parent->parent->colour = red;
       right_rotate( T, x->parent->parent );
     }
    } else {
      . . . /* repeat above with red/black swapped */
```

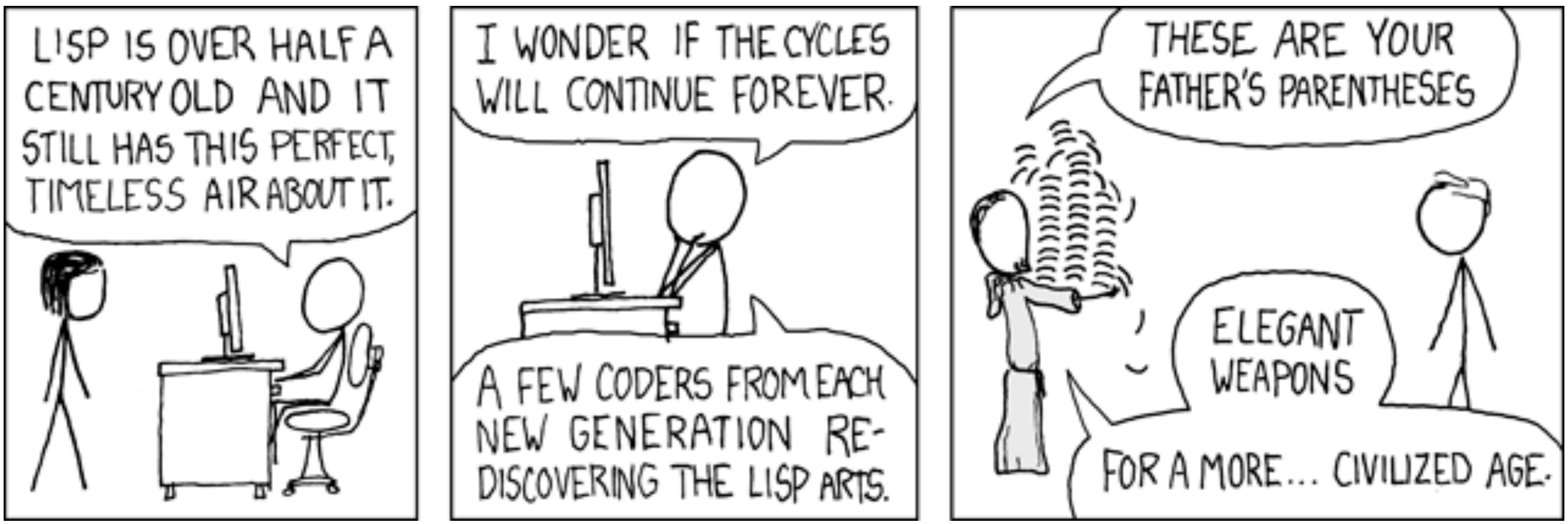
C code (part 2/4)

```
void left_rotate( Tree T, node x ) {
  node y;
  y = x->right;
  x->right = y->left;
  if ( y->left != NULL )
    y->left->parent = x;
  y->parent = x->parent;
  if ( x->parent == NULL )
    T->root = y;
  else if ( x == (x->parent)->left )
    x->parent->left = y;
  else
   x->parent->right = y;
   y->left = x;
   x->parent = y;
}

/* repeat above for right_rotate with “obvious” changes */
```

ML Code for Insert

```
fun balance((Blk,T(Red,T(Red,a,x,b),y,c),z,d)
           |(Blk,T(Red,a,x,T(Red,b,y,c)),z,d)
           |(Blk,a,x,T(Red,T(Red,b,y,c),z,d))
           |(Blk,a,x,T(Red,b,y,T(Red,c,z,d)))) =
                 T(Red,T(Blk,a,x,b),y,T(Blk,c,z,d))
   | balance x = T x

fun ins x Empty = T(R,Empty,x,Empty)
  | ins x (T(color,a,y,b)) =
  if x <= y then balance(color,ins x a,y,b)
  else if x > y then balance(color,a,y,ins x b)
```

XKCD


LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.