Elegance and Replication

Code should be elegant, versatile and minimal

Nice if we can get one method to do the job of 20 !

Especially if it is a fraction of the size of those 20

Achieved by "factoring out" common functionality

Referred to as "DRY" (Don't Repeat Yourself) code...

Some WET code

```
public void processCommand(String action, Unit unit)
{
    if(action.equals("add")) {
        System.out.println("ID of student to add?");
        String id = System.in.readline();
        Student student = cohort.getStudent(id);
        unit.addStudent(student);
    }
    else if(action.equals("remove")) {
        System.out.println("ID of student to remove ?");
        String id = System.in.readline();
        Student student = cohort.getStudent(id);
        unit.removeStudent(student);
    }
}
```

DRYer equivalent

```
System.out.println("ID of student to "+ action +"?");
String id = System.in.readline();
Student student = cohort.getStudent(id);
if(action.equals("add")) unit.addStudent(student);
if(action.equals("remove")) unit.removeStudent(student);
```

DRY Metrics

Various approaches can be used in an attempt to assess the DRYness of code...

- IF density: Large blocks entirely of IF statements
- Line similarity: Similar duplicated lines of code
- Method similarity: "Self plagiarism" of methods

Redundant Code

Whilst we are on the subject of redundant code
What about code that is never actually used at all ?

Happens from time-to-time during evolutionary dev
Trying out some ideas in an experimental method
But never actually linking things in

This is fine, but just be careful not to submit it !
It's easy for checkers to detect this kind of thing ;o)

And Finally: The Eternal Conflict

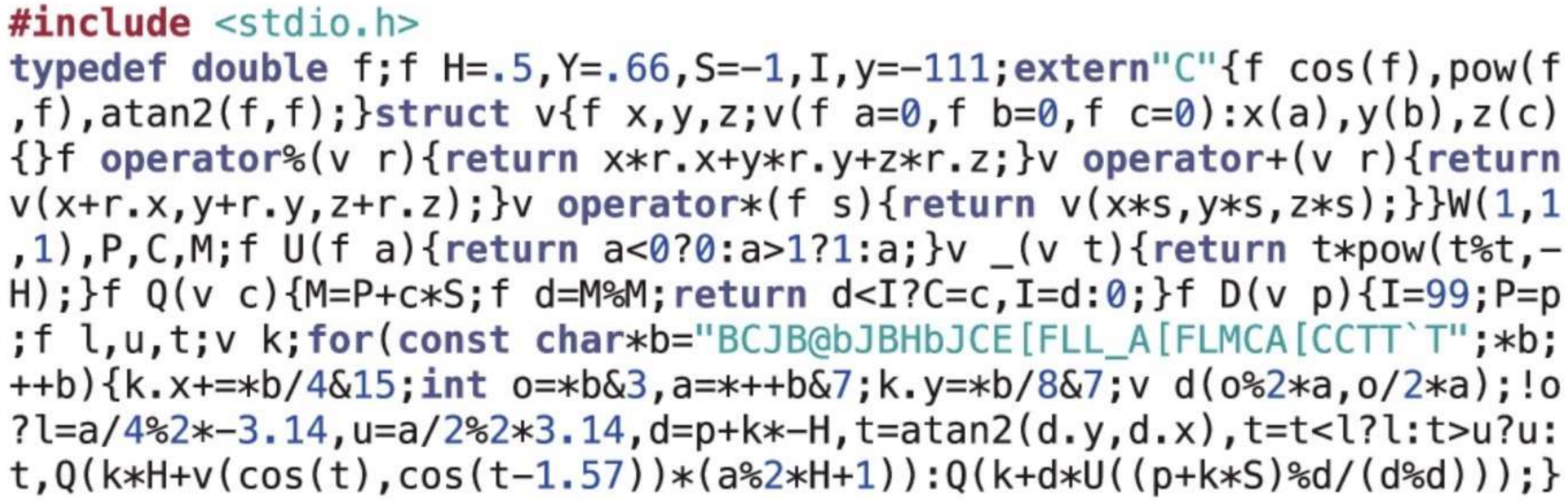
Elegance vs Understandability

There is often a tension between these two

Code can be very compact, clever and efficient...

Yet at the same time totally incomprehensible

(Remember the Ray Tracer ?)



```
#include <stdio.h>
typedef double f;f H=.5,Y=.66,S=-1,I,y=-111;extern"C"{f cos(f),pow(f
,f),atan2(f,f);}struct v{f x,y,z;v(f a=0,f b=0,f c=0):x(a),y(b),z(c)
{}f operator%(v r){return x*r.x+y*r.y+z*r.z;}v operator+(v r){return
v(x+r.x,y+r.y,z+r.z);}v operator*(f s){return v(x*s,y*s,z*s);}}W(1,1
,1),P,C,M;f U(f a){return a<0?0:a>1?1:a;}v _(v t){return t*pow(t%t,-
H);}f Q(v c){M=P+c*S;f d=M%M;return d<I?C=c,I=d:0;}f D(v p){I=99;P=p
;f l,u,t;v k;for(const char*b="BCJB@bJBHbJCE[FLL_A[FLMCA[CCTT`T";*b;
++b){k.x+=*b/4&15;int o=*b&3,a=*++b&7;k.y=*b/8&7;v d(o%2*a,o/2*a);!o
?l=a/4%2*-3.14,u=a/2%2*3.14,d=p+k*-H,t=atan2(d.y,d.x),t=t<l?l:t>u?u:
t,Q(k*H+v(cos(t),cos(t-1.57))*(a%2*H+1)):Q(k+d*U((p+k*S)%d/(d%d)));}
```

Compromise

Sometimes longer, verbose, inelegant code is better !
Future programmers have a chance of understanding

As a Computer Scientist, this can hurt
As a Programmer (or even Developer)...
we know it is the right thing to do