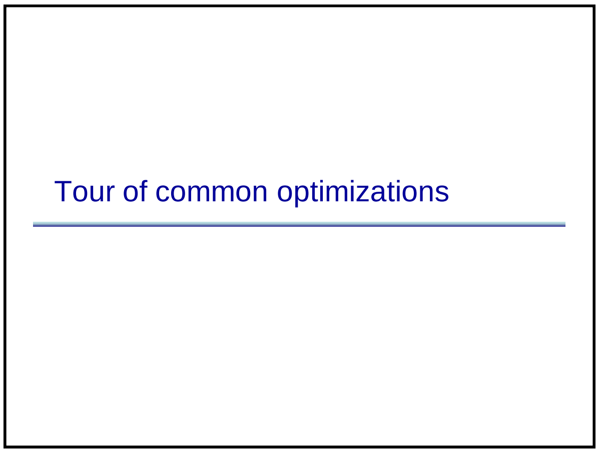
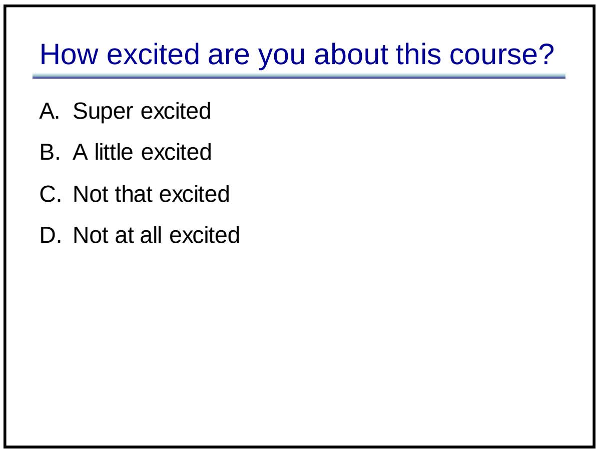


Tour of common optimizations

1



How excited are you about this course?

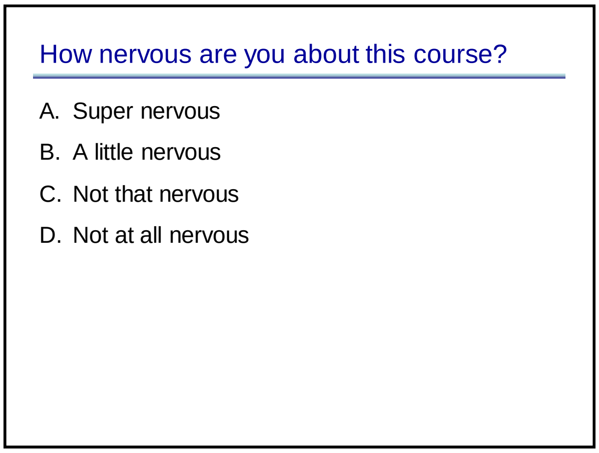
A. Super excited

B. A little excited

C. Not that excited

D. Not at all excited

2



How nervous are you about this course?

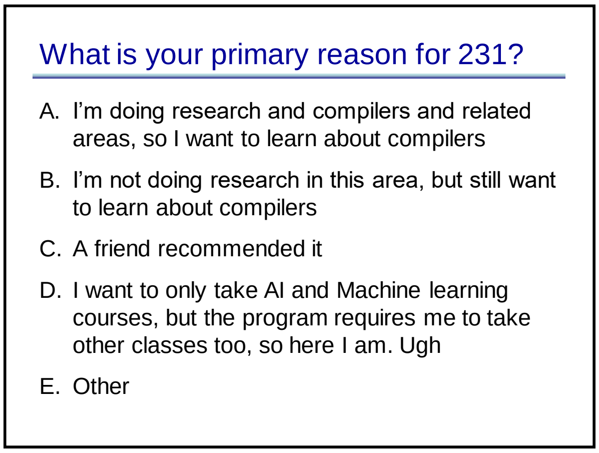
A. Super nervous

B. A little nervous

C. Not that nervous

D. Not at all nervous

3



What is your primary reason for 231?

A. I'm doing research and compilers and related areas, so I want to learn about compilers

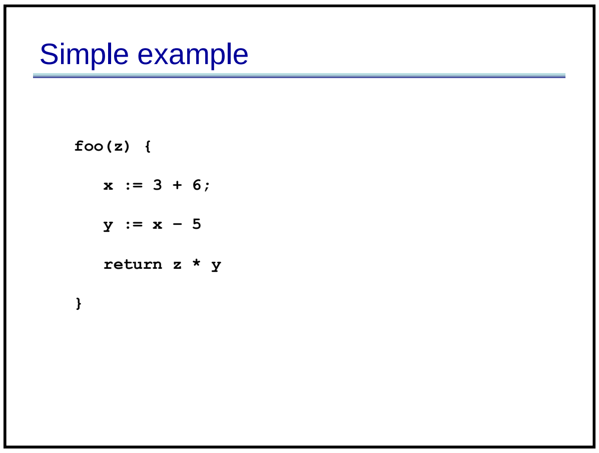
B. I'm not doing research in this area, but still want to learn about compilers

C. A friend recommended it

D. I want to only take AI and Machine learning courses, but the program requires me to take other classes too, so here I am. Ugh

E. Other

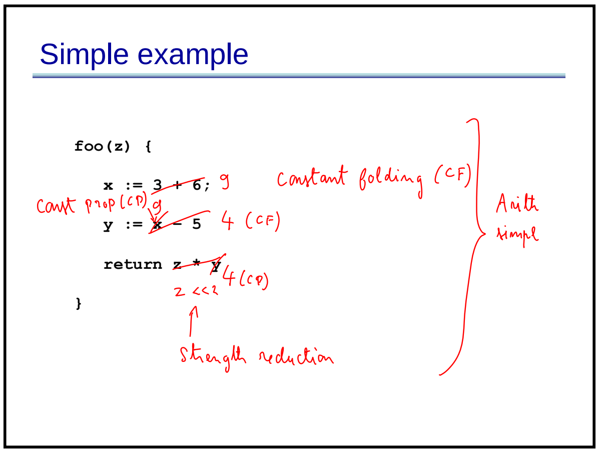
4



Simple example

```
foo(z) {
   x := 3 + 6;
   y := x - 5
   return z * y
}
```

5



Simple example

foo(z) {

x := ~~3 + 6~~; 9   Constant folding (CF)

Const prop (CP) 9

y := ~~x - 5~~ 4 (CF)

return ~~z * y~~ 4 (CP)

z << 2

}

Strength reduction

Arith simpl

6

Another example

```
x := a + b;

...

y := a + b;
```

7

Another example

```
x := a + b;

...

y := ~~a + b~~; x
```

} only if x, a, b not modified!

8

Another example

```
if (...) {
   a := read();
   x := a + b;
   print(x);
}

...

y := a + b;
```

9

Another example

```
if (...) {
   a := read(); t := a+b
   x := ~~a + b~~; t
   print(x);
} else { t := a+b }

...

y := ~~a + b~~; t
```

Partial Redundancy Elimination PRE

10

Another example

```
x := y
...
z := z + x
```

11

Another example

```
x := y
...
z := z + ~~x~~ y
```

} x,y not modified

copy prop

12

Another example

```
x := y
...
z := z + y
```

What if we run CSE now?

13

Another example

```
x := y
...
z := z + ~~y~~ x
```

What if we run CSE now?

14

Another example

```
x := y**z

...

x := ...
```

15

Another example

```
~~x := y**z~~

...

x := ...
```

} if x is not used

dead assignment elim
(unused assignment elim)

- Often used as a clean-up pass

$$\begin{tikzcd} \texttt{x := y} \\ \texttt{z := z + x} \arrow[r, "\text{Copy prop}"] & \texttt{x := y} \\ \texttt{z := z + y} \arrow[r, "\text{DAE}"] & \cancel{\texttt{x := y}} \\ \texttt{z := z + y} \end{tikzcd}$$

16

Another example

```
if (false) {

   ...

}
```

17

Another example

```
~~if (false) {~~

   ~~...~~

~~}~~
```

dead code elim
(unreachable code elim)

Another common clean up opt

18

Another example

- In Java:

```
a = new int [10];
for (index = 0; index < 10; index ++) {
   a[index] = 100;
}
```

19

Another example

- In "lowered" Java:

```
a = new int [10];
for (index = 0; index < 10; index ++) {
   if (index < 0 || index >= a.length()) {
      throw OutOfBoundsException;
   }
   a[index] = 0;
}
```

20

Another example

- In "lowered" Java:

```
a = new int [10];  ①
for (index = 0; index < 10; index ++) {
   if (index < 0 || index >= a.length()) {
      throw OutOfBoundsException;
   }
   a[index] = 0;
}
```

~~if (index < 0 || index >= a.length()) { throw OutOfBoundsException; }~~ ← Branch folding + unreachable code elim

~~a.length()~~ 10 ← Kinda like CP if we assume stmt ① acts like a.length:=10

index ∈ [0..9] ← Range analysis

21

Another example

```
p := &x;
*p := 5
y := x + 1;
```

22

Another example

```
p := &x;
*p := 5
y := x + 1;
```

× ~~*p~~ := 5

y := ~~x + 1;~~ 6

~~x~~ 5

pointer / alias analysis

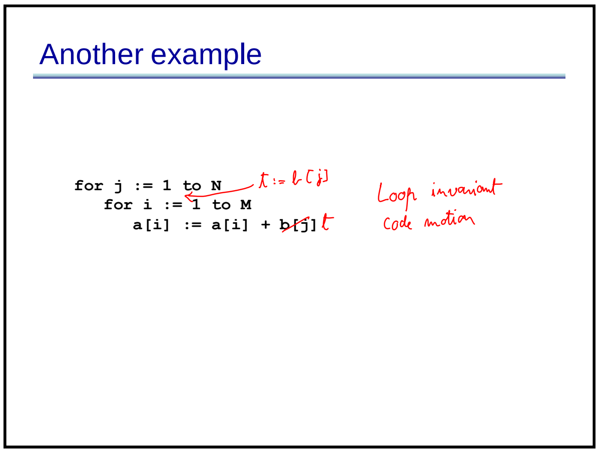
```
x := 5;
*p := 3
y := x + 1;  ⇨  ???
```

23

Another example

```
for j := 1 to N
   for i := 1 to M
      a[i] := a[i] + b[j]
```

24

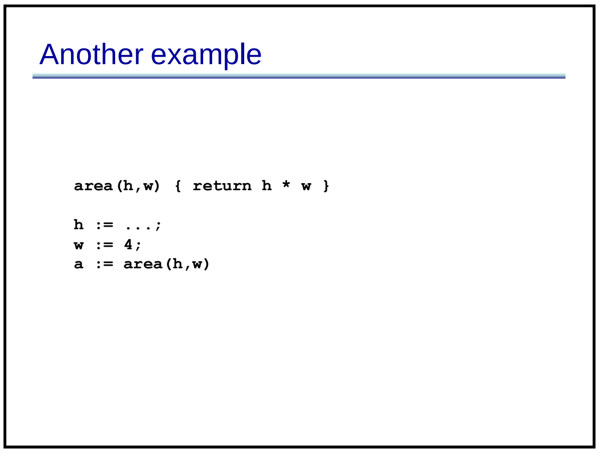


Another example

```
for j := 1 to N            t := b[j]
   for i := 1 to M
      a[i] := a[i] + ~~b[j]~~ t
```

Loop invariant code motion

25

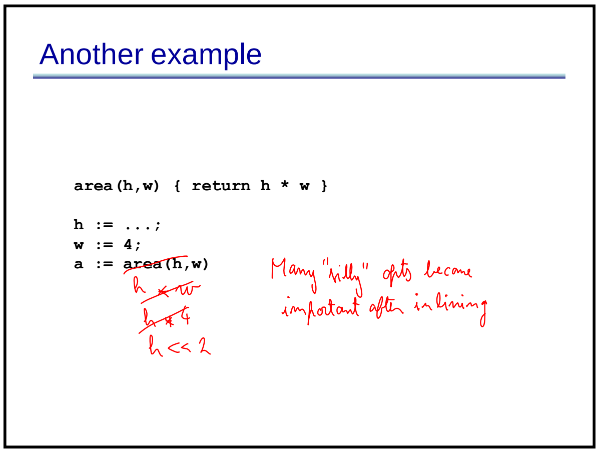


Another example

```
area(h,w) { return h * w }

h := ...;
w := 4;
a := area(h,w)
```

26



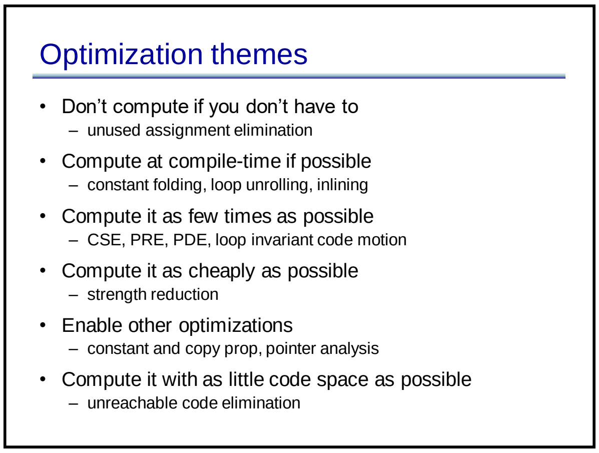
Another example

```
area(h,w) { return h * w }

h := ...;
w := 4;
a := ~~area(h,w)~~
       ~~h * w~~
        ~~h * 4~~
         h << 2
```

Many "silly" opts become important after inlining

27



Optimization themes

- Don't compute if you don't have to
  - unused assignment elimination
- Compute at compile-time if possible
  - constant folding, loop unrolling, inlining
- Compute it as few times as possible
  - CSE, PRE, PDE, loop invariant code motion
- Compute it as cheaply as possible
  - strength reduction
- Enable other optimizations
  - constant and copy prop, pointer analysis
- Compute it with as little code space as possible
  - unreachable code elimination

28