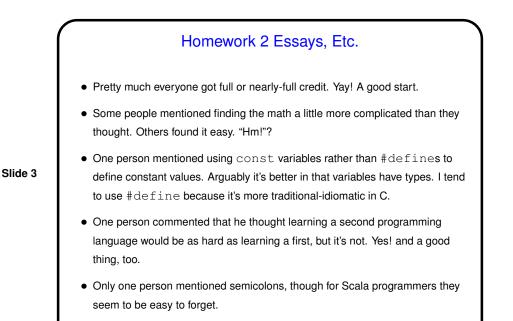
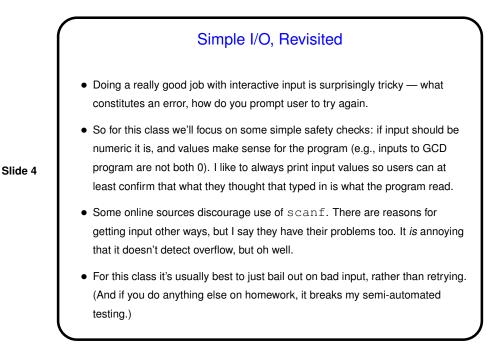
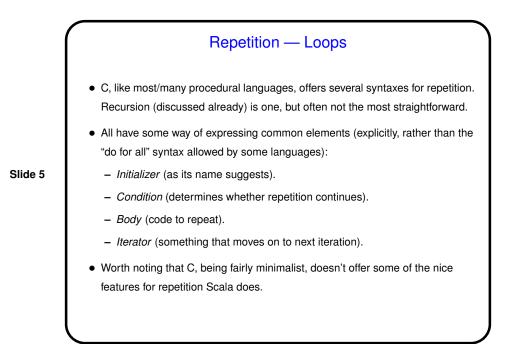


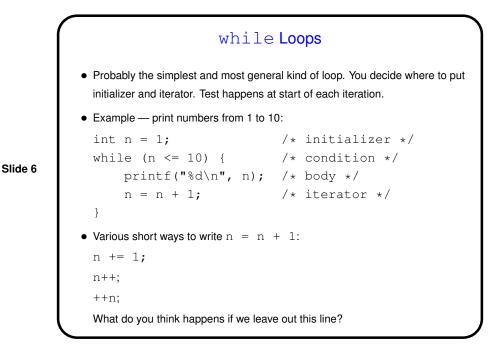
## Minute Essay From Last Lecture

 Responses to the question about recursion varied — some people felt comfortable with it, others had learned it but forgotten a lot, and it was new to the few who haven't taken CS1. The recursion needed for Homework 3 is a simple form so I hope will not (did not?) stress anyone.

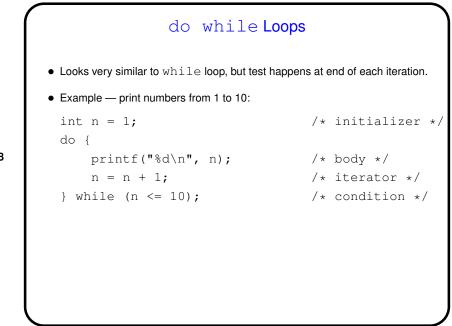


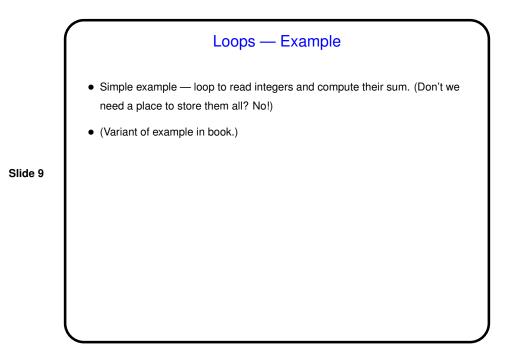


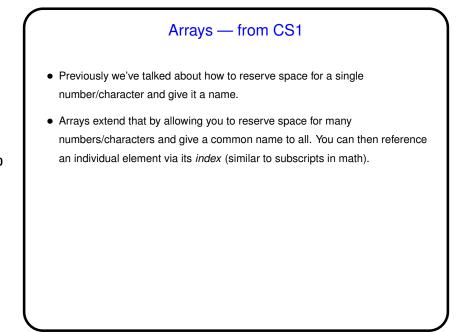




for Loops
• Probably the most common type of loop. Particularly useful for anything
involving counting, but can be more general. Syntax has explicit places for
initializer, condition, iterator (so it's less likely you'll forget one of them).
• Example — print numbers from 1 to 10:
for (int n = 1; n <= 10; ++n) {
 printf("%d\n", n);
}
• Initializer happens once (at start); condition is evaluated at the start of each
iteration; iterator is executed at the end of each iteration. (Note that C89
standard required that n be declared outside the loop.)</pre>







## Arrays in C

• Declaring an array — give its type, name, and how many elements. Examples:

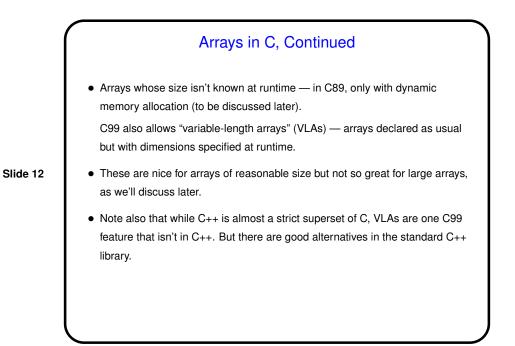
```
int nums[10];
double stuff[N];
```

Slide 11

(The second example assumes N is declared and given a value previously. In C89, it had to be a constant. In C99, it can be a variable — "variable-length array".)

• Alternatively, give "initializer" (list of values) and let compiler figure out size. Example:

```
int nums[] = { 2, 4, 6, 8 };
```



## Arrays in C, Continued

• Referencing an array element — give the array name and an index (ranging from 0 to array size minus 1). Index can be a constant or a variable. Then use as you would any other variable. Examples:

```
nums[0] = 20;
printf("%d\n", nums[0]);
```

(Notice that the second example passes an array element to a function. AOK!)

• So far nothing new, just different syntax. But ...

