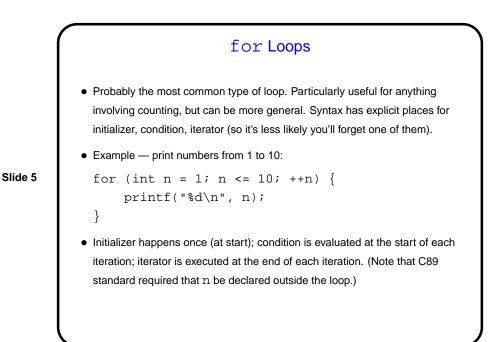
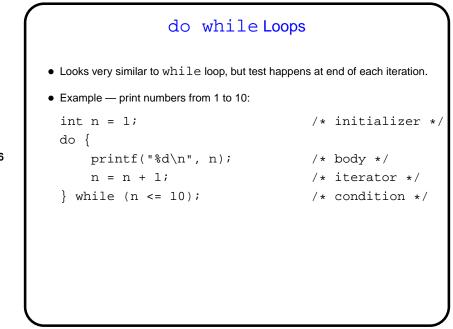
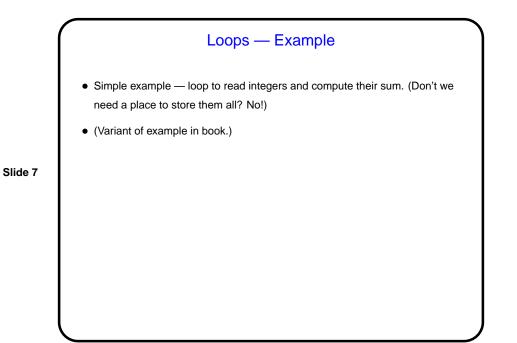
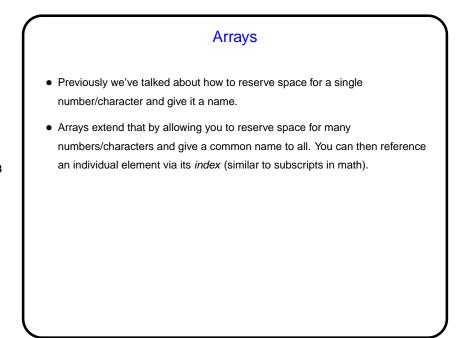


	while Loops
	 Probably the simplest kind of loop. You decide where to put initializer and iterator. Test happens at start of each iteration.
Slide 4	• Example — print numbers from 1 to 10:
	<pre>int n = 1; /* initializer */</pre>
	while (n <= 10) {
	printf("%d\n", n); /* body */
	n = n + 1;
	}
	• Various short ways to write $n = n + 1$:
	n += 1;
	n++;
	++n;
	What do you think happens if we leave out this line?





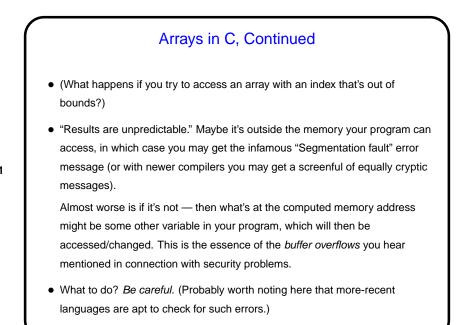




```
Arrays in C
• Declaring an array — give its type, name, and how many elements.
Examples:
    int nums[10];
    double stuff[N];
    (The second example assumes N is declared and given a value previously. In
    C89, it had to be a constant. In C99, it can be a variable.)
• Referencing an array element — 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!)
```

Slide 10

Arrays in C, Continued We said if you declare an array to be of size *n* you can reference elements with indices 0 through *n* – 1. What happens if you reference element -1? *n*? 2*n*? Well, the compiler won't complain. At runtime, the computer will happily compute a memory address based on the starting point of the array and the index. If the index is "in range", all is well. If it's not (i.e., it's "out of bounds) ...



Arrays — Example
Back story: Conventional wisdom says that many library functions for generating sequences of random numbers aren't very random in their least-significant bits, so mapping their output to a small range using the mod operator isn't a good idea.
Slide 12
We could write a short program to check, in a crude way, whether that's true, or at least how well the results are distributed over the range: Prompt for how many "random" numbers to generate and for a divisor, then generate the sequence, divide each by the divisor, and count how many have remainder 0, remainder 1, etc.

