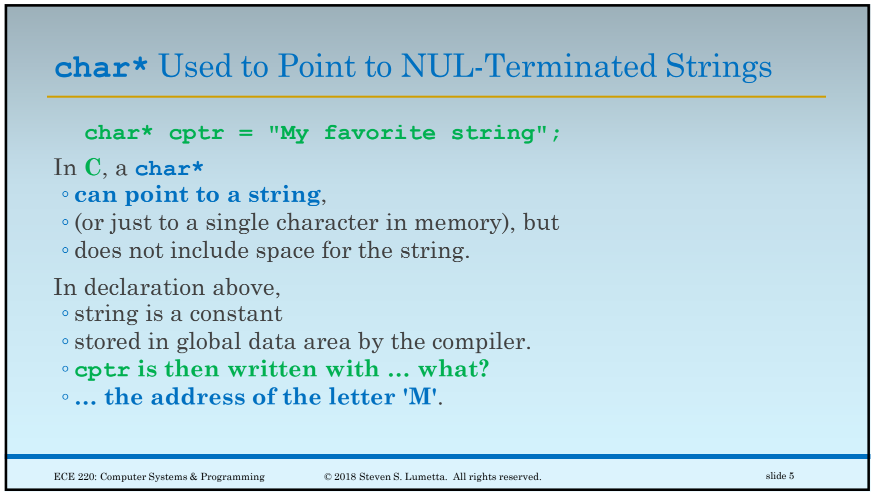
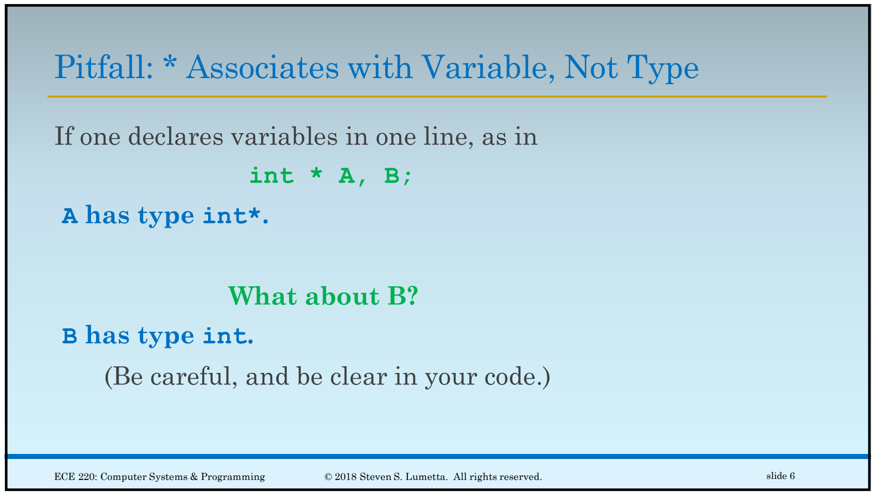


`char*` Used to Point to NUL-Terminated Strings

```
char* cptr = "My favorite string";
```

In **C**, a **`char*`**
- **can point to a string**,
- (or just to a single character in memory), but
- does not include space for the string.

In declaration above,
- string is a constant
- stored in global data area by the compiler.
- **`cptr` is then written with … what?**
- **… the address of the letter 'M'**.



Pitfall: * Associates with Variable, Not Type

If one declares variables in one line, as in

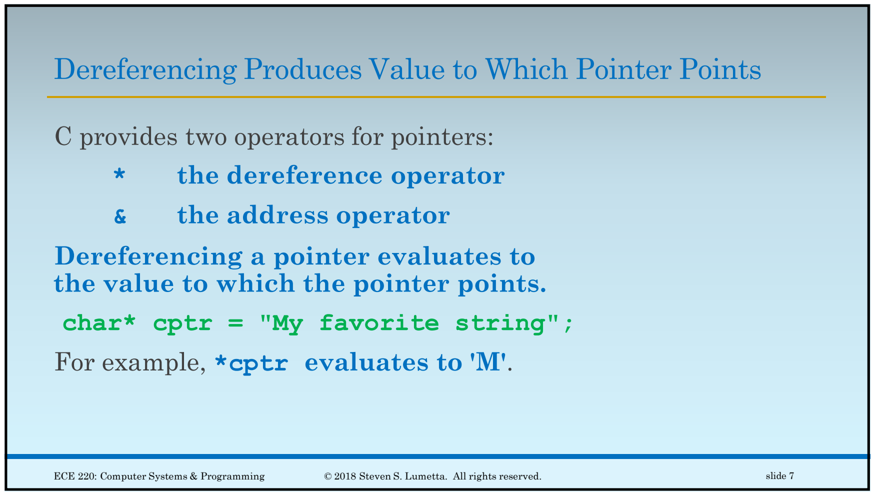
```
int * A, B;
```

`A` has type `int*`.

What about B?

`B` has type `int`.

(Be careful, and be clear in your code.)



Dereferencing Produces Value to Which Pointer Points

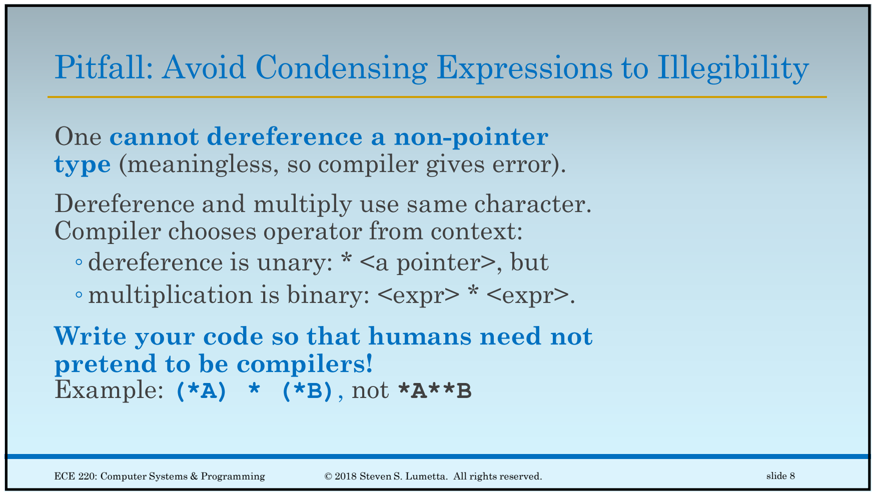
C provides two operators for pointers:

`*`	**the dereference operator**
`&`	**the address operator**

Dereferencing a pointer evaluates to the value to which the pointer points.

```
char* cptr = "My favorite string";
```

For example, **`*cptr` evaluates to 'M'**.



Pitfall: Avoid Condensing Expressions to Illegibility

One **cannot dereference a non-pointer type** (meaningless, so compiler gives error).

Dereference and multiply use same character. Compiler chooses operator from context:
- dereference is unary: * <a pointer>, but
- multiplication is binary: <expr> * <expr>.

Write your code so that humans need not pretend to be compilers!
Example: **`(*A) * (*B)`**, not **`*A**B`**