

Slide 1

User-Defined Types

 So far we've only talked about representing very simple types — numbers, characters, text strings, arrays, and pointers. A lot can be done with just these types, but for a lot of applications it's really useful to be able to represent more-complex kinds of data, and in general it's useful to be able to define your own types.

Slide 2

• (Aside: A standard computer-science definition of "type" is "a set of values plus some operations on them". So for example C's int represents a set of integer values in a fixed range, with operations including arithmetic and relational operators.)

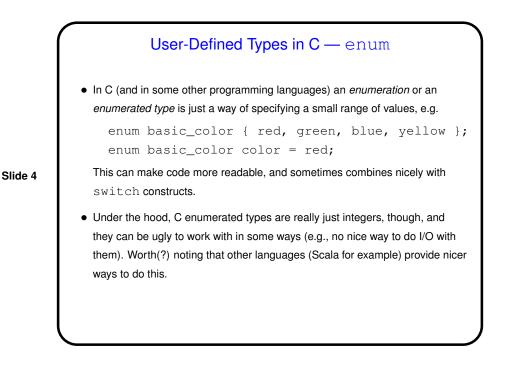


• typedef just provides a way to give a new name to an existing type, e.g.:

```
typedef charptr char *;
```

• This can make your code more readable, or allow you to isolate things that might be different on different platforms (e.g., whether to use float or double in some application) in a single place.

Slide 3



User-Defined Types in C — struct

It's also useful to have ways of representing more-complex "types".
 Simple examples include rational numbers (with integer numerator and denominator) and points in 2D or 3D space (with two or three coordinates).
 More-complex examples — well, it's almost "the sky's the limit", but as one example think about what might be involved in writing a program with a GUI — it would seem to make sense to have some way of representing "windows" and "buttons" and so forth.

Slide 5

• "Object-oriented" languages provide a nice way to do this. C doesn't provide a really nice way to do this, but it does provide a way, via structs.

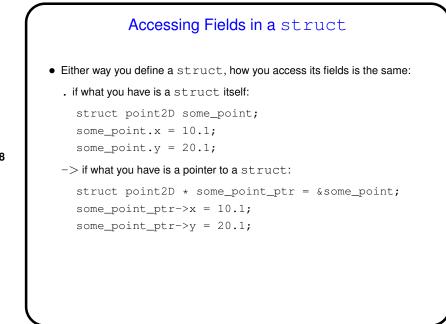
structs in C

• An array in C (and in most if not all other programming languages) is a collection of data items all of the same type, and you reference individual items via indices. Arrays can have any number of elements. You define an array by giving its type and its dimensions.

Slide 6

• In contrast, a struct is a collection of data items of possibly different different types. Individual items are called "fields" and have names by which you can refer to them. For a given struct type, all objects of that type have the same fields. You define a struct by giving names and types of its fields. Two syntaxes for doing that.

	Defining a struct	
Slide 7	 One way to define uses typedef: typedef struct { double x; double y; } point2D; point2D some_point; Another way doesn't: struct point2D { double x; double y; }; struct point2D some_point; 	



Slide 8

