A Structure Definition Can Be Used as a Type

A structure definition (without a semicolon)
- **`struct { … }`**
- **is also a type**.
- Such a type has no name.

But it can be used to declare variables:

```
struct { … } my_structure;
```

And it can be given a name:

```
typedef struct { … } my_type_t;
```

 © 2018 Steven S. Lumetta. All rights reserved.

Structure Definition and `typedef` can be Merged

You may sometimes see a named structure definition merged with a **`typedef`**:

```
typedef struct player_t {
    ...
} Player;
```

With the form above,
- **`player_t*`** cannot be used in the structure definition;
- instead, use **`struct player_t*`**.
- And the two cannot be split, of course.

 © 2018 Steven S. Lumetta. All rights reserved.

Pitfall: Saving Typing at the Expense of Code Readability

Do not define types
- **to save a little typing**
- **at the expense of clear code.**

For example,

```
typedef int* Int;
```

What's the problem?

```
some_function (x, y);
```

Do x and y change? Maybe…*

*The pointers do not change, of course, but the data are the things to which the pointers point.

 © 2018 Steven S. Lumetta. All rights reserved.

Enumerating Possibilities with `enum` is Also Useful

Another useful type: enumerations.

What's an enumeration?
1. **a list of things**
2. **with some common feature**
3. **numbered consecutively.**

 © 2018 Steven S. Lumetta. All rights reserved.