University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Defining New Types

Use `typedef` to Define New Types

It gets a bit tiresome to keep writing **`struct`** everywhere.

Instead, we can create new types by writing

```
typedef <base type> <list of types>;
```

The **`typedef`** statement **looks just like a variable declaration**,

- except that it **starts with `typedef`**, and
- **new types are defined** (instead of variables).

Typical Uses of `typedef`

The most common forms are…

```
typedef struct player_t Player;
```

or

```
typedef struct player_t player_t;
```

Note that

1. the **same name can be used** (without **`struct`**), and
2. the **structure definition need not appear before** these definitions.

Information Hiding: Split Type and Structure Definitions

Only functions that

- implement operations on a data structure
- need the structure definition.

To **hide the implementation, a header file can include**

```
typedef struct player_t player_t;
```

- to **allow other code to use `player_t*`**,
- while the **structure definition and all operations are in a single file**.

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;
```


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.

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.

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.**

Enumerations Start with 0 in C

In **C**, **enumerations**
- **start with 0**, but
- **can be overridden**.

For example, given

```
enum {FALSE, TRUE};
```

- **FALSE has value 0**, and
- **TRUE has value 1**.

Values are Numbered (and Re-Numbered) Automatically

We **count the number of values by adding an extra name** at the end:

```
enum {
        SPACE_EMPTY,
        SPACE_FULL,
        SPACE_BLOCK,
        NUM_SPACE_TYPES
};
```

If new names are added,
- **NUM_SPACE_TYPES grows automatically**
- along with any arrays based on it.

Enumerations Can Also Be Used for Bit Vectors

Enumerations can also be used **to name bits in a bit vector**:

```
enum {
    LEFT_WALL  = 1,
    RIGHT_WALL = 2,
    UPPER_WALL = 4,
    LOWER_WALL = 8,
    HAS_EXIT   = 16
};
```

Notice how the default values can be overridden.

