Enumerated Types

An Example

```
#include <stdio.h>

// our first type declaration
enum day {MON, TUE, WED, THU, FRI, SAT, SUN};

int main( void ) {
        enum day d;
        d = MON; printf( "MON=%d\n", d );
        d++; printf( "MON++ (TUE)=%d\n", d );
        d = SUN; printf( "SUN=%d\n", d );
        d--; printf( "SUN-- (SAT)=%d\n", d );
        return 0;
}
```

Enums as integers

Enum values are basically integer constants

By default, they start at 0 and increment by 1. So MON is 0, TUE is 1, up to SUN being 6

The new type is called `enum day` not `day`, so the variable declaration is `enum day d`

An `enum day` variable such as `d` can be set to either a day value, or to a small int in the range 0..6, then `d` can be incremented or decremented - just like an ordinary int variable. But as our days are (internally) represented by ints in the range 0..6, you shouldn't set `d = SUN` and then `d++`

Similarly you shouldn't assign a value out of range, as in d = 17

You should still try to keep your enums "valid" yourself by checks

Specifying enum constants

You can set the enum values explicitly

Suppose we want the internal range of days to be from 10..16 (i.e. MON being 10, TUE being 11 etc)

To do this, change the enum declaration to read: `enum day {MON=10, TUE, WED, THU, FRI, SAT, SUN}`

`enum flags {AMBIENT=1, DIFFUSE=2, SPECULAR=4}`

This enumeration has holes in it; therefore, it is bad practice

You can loop across all, or a subrange of, enums as long as their internal values are contiguous:

```
for( enum day d = MON; d<=SUN; d++ ) {
        ...
}
```

You can select on an enum:

```
switch( d ) {
case MON: printf( "Monday" ); break;
case TUE: printf( "Tuesday" ); break;
case WED: printf( "Wednesday" ); break;
case THU: printf( "Thursday" ); break;
case FRI: printf( "Friday" ); break;
case SAT: printf( "Saturday" ); break;
case SUN: printf( "Sunday" ); break;
}
```

Typedef

Personally I don’t like multi-word typenames such as `unsigned int` or `enum day`

We can avoid this by using a new C keyword `typedef` which allows you to create a new named type

If we rewrite our `enum day` example as `typedef enum {MON, TUE, WED, THU, FRI, SAT, SUN} day;`

This declares an anonymous enumeration (no enumeration name after enum), containing those 7 values, and then names the whole type `day`

Now we declare our variable d as `day d`

`typedef TYPE TYPENAME`