band = blue; // valid, blue is an enumerator
band = 2000; // invalid, 2000 not an enumerator

Thus, a spectrum variable is limited to just eight possible values. Some compilers issue a compiler error if you attempt to assign an invalid value, whereas others issue a warning. For maximum portability, you should regard assigning a non-enum value to an enum variable as an error.

Only the assignment operator is defined for enumerations. In particular, arithmetic operations are not defined:

```c
band = orange; // valid
++band; // not valid
band = orange + red; // not valid
...
```

However, some implementations do not honor this restriction. That can make it possible to violate the type limits. For example, if `band` has the value ultraviolet, or 7, then `++band`, if valid, increments `band` to 8, which is not a valid value for a spectrum type. Again, for maximum portability, you should adopt the stricter limitations.

Enumerators are of integer type and can be promoted to type `int`, but `int` types are not converted automatically to the enumeration type:

```c
int color = blue; // valid, spectrum type promoted to int
band = 3; // invalid, int not converted to spectrum
color = 3 + red; // valid, red converted to int
...
```

Note that even though 3 corresponds to the enumerator `green`, assigning 3 to `band` is a type error. But assigning `green` to `band` is fine, for they both are type `spectrum`. Again, some implementations do not enforce this restriction. In the expression `3 + red`, addition isn't defined for enumerators. However, `red` is converted to type `int`, and the result is type `int`. Because of the conversion from enumeration to `int` in this situation, you can use enumerations in arithmetic expressions combining them with ordinary integers even though arithmetic isn't defined for enumerations themselves.

You can assign an `int` value to an `enum` provided that the value is valid and that you use