Lecture 02.03 Exploring C—compile-time integral constants

Often, we want to define a symbol that has a single *integral value*—an integer—throughout our program. Fortunately, C lets us do that many ways. Unfortunately, it can be hard to choose among them. **integral value**

The primary ways are `#define`s (macros), **`enum`**s (enumerations), and **`const int`**s. When choosing among them, our primary concerns are code readability, debuggability, and compile-time optimization.

Box 02.1 `static` is not constant

Several times thus far, including in the listing of Figure 00.6, we have declared **`static int`**s. These are variables *not* constants, as their name might suggest. Rather, they retain their value between function calls—but that value can be changed within the function (and the new value retained).

The last of these means a compiler (or preprocessor before the compiler) can replace each instance of the symbol with its constant value (since it never chances). There are subtle differences in how each compiler works, but most of the time all three of our options yield replaced compile-time constants. However, `#define`s are the best guarantee (because it actually happens before compilation, via *preprocessing*), **`enum`**s a close second, and **`const int`**s a respectable third. **preprocessing**

In terms of debuggability, the rankings are probably best reversed; that is, in decreasing debuggability: **`const int`**s, **`enum`**s, and `#define`s. Macros (`#define`s) are most difficult because the compiler can't usually give useful error codes related to them (since the compiler typically knows nothing of them due to preprocessing).

Readability is rather subjective, but **`enum`**s are typically considered strong in this regard, especially with its automatic enumeration of symbols.

A way to demonstrate this is to show the same example, written these three ways. Let's define an integral value to each day of the week, then write a script that prints a value.

```
#include <stdio.h>
enum day {
    sunday, monday, tuesday, wednesday,
    thursday, friday, saturday
};
```

```
enum day today = monday;
enum day checkout = friday;

int main() {
    printf("Checkout in %d days.", checkout-today);
    return 0;
}
```

```
Checkout in 4 days.
```

```
#include <stdio.h>
#define sunday 0
#define monday 1
#define tuesday 2
#define wednesday 3
#define thursday 4
#define friday 5
#define saturday 6
#define today monday
#define checkout friday

int main() {
    printf("Checkout in %d days.", checkout-today);
    return 0;
}
```

```
Checkout in 4 days.
```

```
#include <stdio.h>
const int sunday = 0;
const int monday = 1;
const int tuesday = 2;
const int wednesday = 3;
const int thursday = 4;
const int friday = 5;
const int saturday = 6;
const int today = monday;
const int checkout = friday;

int main() {
    printf("Checkout in %d days.", checkout-today);
    return 0;
}
```

```
Checkout in 4 days.
```

Preference among these three options is hotly debated, but it seems `enum`s are the most readable and the "just right" option in terms of reliable compile-time integral constant replacement and debuggability.

It is important to remember that `#define`s can be used for much more than integer replacement: function-like macros, for instance, are very useful.